Deutsch

Fujitsu Software BS2000

openUTM V7.0

Anwendungen programmieren mit KDCS für COBOL, C und C++

Benutzerhandbuch

Stand der Beschreibung:
openUTM V7.0A00

Ausgabe November 2019

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an bs2000.info@fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2015

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001: 2015 erfüllt.

Copyright und Handelsmarken

Copyright © 2025 Fujitsu

Alle Rechte vorbehalten.
Liefermöglichkeiten und technische Änderungen vorbehalten.

Alle verwendeten Hard- und Softwarenamen sind Handelsnamen und/oder Warenzeichen der jeweiligen Hersteller.

Inhaltsverzeichnis

Anwendungen Programmieren mit KDCS für COBOL, C und C++ 10
1 Einleitung 11
1.1 Zielgruppe und Konzept des Handbuchs 13
1.2 Wegweiser durch die Dokumentation zu openUTM 14
1.2.1 openUTM-Dokumentation 15
1.2.2 Dokumentation zum openSEAS-Produktumfeld 18
1.2.3 Readme-Dateien 19
1.3 Änderungen in openUTM V7.0 20
1.3.1 Neue Server-Funktionen 21
1.3.2 Entfallene Server-Funktionen 26
1.3.3 Neue Client-Funktionen 27
1.3.4 Neue Funktionen für openUTM WinAdmin 28
1.3.5 Neue Funktionen für openUTM WebAdmin 29
1.4 Darstellungsmittel 30
2 Aufbau und Einsatz von UTM-Programmen 32
2.1 Vorgangs-Konzept von openUTM 35
2.2 Aufbau eines Teilprogramms 38
2.2.1 Programmrahmen 39
2.2.2 Aufbau eines Dialog-Teilprogramms 41
2.2.3 Reentrant-Fähigkeit von Teilprogrammen 43
2.3 Strukturierung von Vorgängen 44
2.3.1 Mehrschritt-Vorgänge 45
2.3.2 Mehrere Teilprogramme in einem Verarbeitungsschritt 48
2.3.3 Mehrere Verarbeitungsschritte in einem Teilprogramm 50
2.3.4 Unterprogramm-Aufrufe aus Teilprogrammen 52
2.3.5 Vorgänge ketten 53
2.3.6 Vorgänge kellern 54
2.4 Message Queuing (Asynchron-Verarbeitung) 55
2.4.1 Dead Letter Queue 56
2.4.2 Nachrichten an UTM-gesteuerte Queues 57
2.4.2.1 Ausgabe-Aufträge 58
2.4.2.2 Hintergrund-Aufträge 59
2.4.2.3 MQ-Aufrufe der KDCS-Schnittstelle 60
2.4.2.4 Aufbau eines Asynchron-Vorgangs 61
2.4.2.5 Redelivery bei Hintergrundaufträgen 68
2.4.3 Nachrichten an Service-gesteuerte Queues 69
2.4.3.1 USER-Queues 70

2.4.3.2 TAC-Queues 71
2.4.3.3 Temporäre Queues 72
2.4.3.4 MQ-Aufrufe der KDCS-Schnittstelle 73
2.4.3.5 Lebensdauer von Queues und Queue-Nachrichten 74
2.4.3.6 Löschen von USER- und TAC-Queues mittels programmierter Administration 75
2.4.3.7 Beispiele 76
2.5 Die KDCS-Speicherbereiche bei openUTM 83
2.5.1 Standard Primärer Arbeitsbereich (SPAB) 87
2.5.2 Kommunikationsbereich (KB) 89
2.5.3 Lokaler Sekundärer Speicherbereich (LSSB) 91
2.5.4 Globaler Sekundärer Speicherbereich (GSSB) 92
2.5.5 Terminal-spezifischer Langzeitspeicher (TLS) 93
2.5.6 User-spezifischer Langzeitspeicher (ULS) 94
2.5.7 Benutzer-Protokolldatei 95
2.5.8 Weitere Bereiche 96
2.5.9 Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB) 97
2.6 Programmierung von Fehlerroutinen 98
2.7 Teilnachrichten 99
2.8 Kommunikationspartner einer UTM-Anwendung 101
2.9 Ausgaben auf Drucker 102
2.9.1 Hardcopy-Betrieb mit openUTM auf BS2000-Systemen 103
2.9.2 Druckaufträge 104
2.10 Unterstützung von Ausweislesern auf BS2000-Systemen 106
2.10.1 Anmelden an die Anwendung mit Ausweisleser (BS2000-Systeme) 107
2.10.2 Dateneingabe per Ausweis (BS2000-Systeme) 108
3 Zusammenarbeit mit Datenbanken 109
3.1 UTM-Transaktion und DB-Transaktionen 111
3.2 Programmierung von ESQL-Teilprogrammen 113
3.3 Fehlerbehandlung bei Datenbank-Kopplung 114
4 Bildschirmfunktionen 115
4.1 Einsatz von Formaten in openUTM auf BS2000-Systemen 116
4.1.1 Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme) 119
4.1.2 Vorgänge über Basisformate starten (BS2000-Systeme) 120
4.1.3 Einsatz von Teilformaten (BS2000-Systeme) 122
4.1.3.1 Ausgabeformatierung mit mehreren Teilformaten (BS2000-Systeme) . . . 123
4.1.3.2 Eingabeformatierung mit mehreren Teilformaten (BS2000-Systeme) 124
4.1.4 Nachrichtenfluss bei formatierten Nachrichten (BS2000-Systeme) 126
4.1.5 Ausgaben auf Drucker im Formatmodus (BS2000-Systeme) 127
4.2 Beeinflussen der Ausgabe im Zeilenmodus (BS2000- Systeme) 128
4.3 Ausgaben auf Drucker im Zeilenmodus 130

4.4 Bildschirmwiederanlauf 131
4.5 Formatnamen beim Nachrichtenaustausch mit UPIC-Clients 132
5 Programmaufbau bei verteilter Verarbeitung 133
5.1 Adressierung ferner Vorgänge 134
5.2 Verteilte Dialoge 136
5.2.1 Steuern der Kommunikation im Programm 137
5.2.2 Fehlerbehandlung im Teilprogramm 138
5.2.2.1 Programmiertes Rücksetzen 139
5.2.2.2 Fehlerbehandlung nach Vorgangs-Wiederanlauf 142
5.2.3 Lastverteilung über LPAP-Bündel 144
5.3 Verteilte Dialoge über LU6.1 145
5.3.1 Programmierhilfen 146
5.3.2 Programmierregeln und Empfehlungen 148
5.3.3 Bestehende Teilprogramme als LU6.1-Auftragnehmer 156
5.3.4 Beispiel: verteilter Dialog über LU6.1 157
5.4 Verteilte Dialoge über OSI TP 161
5.4.1 Funktionseinheiten 162
5.4.2 Programmierhilfen 164
5.4.3 Programmierregeln für Dialoge ohne Functional Unit Commit 169
5.4.4 Programmierregeln für Dialoge mit Functional Unit Commit 170
5.4.5 Programmierregeln bei der Kommunikation mit BeanConnect 173
5.4.6 Besonderheiten bei Rollback und Wiederanlauf 174
5.4.7 Nutzung bestehender Teilprogramme für OSI TP-Kommunikation 176
5.4.8 Besonderheiten bei heterogener Kopplung 178
5.4.9 Beispiele: verteilte Dialoge über OSI TP 179
5.4.9.1 Ein Auftragnehmer 180
5.4.9.2 Mehrere Auftragnehmer 190
5.4.9.3 Komplexere Dialogbäume 192
5.4.9.4 Beendigung eines Auftragnehmers mit CTRL AB 198
5.5 UTM-gesteuerte Queues bei verteilter Verarbeitung 199
5.5.1 Auftraggeber-Seite 200
5.5.2 Auftragnehmer-Seite 201
5.6 Service-gesteuerte Queues bei verteilter Verarbeitung 202
6 Programmaufbau bei Kommunikation mit Transportsystem-Anwendungen . . 203
6.1 Kommunikation mit TS-Anwendungen vom Typ APPLI 204
6.2 Kommunikation mit Socket-USP-Anwendungen 205
6.2.1 Eingabe-Nachrichten für openUTM 206
6.2.2 Ausgabe-Nachrichten von openUTM 207
6.2.3 Aufbau des USP-Headers 208
7 KDCS-Aufrufe 210
7.1 Übersicht über alle KDCS-Aufrufe 211

7.2 Hinweise zur Beschreibung der KDCS-Aufrufe ... 215
7.3 APRO Adressieren eines Auftragnehmer-Vorgangs ... 216
7.4 CTRL Steuern eines OSI TP-Dialogs ... 225
7.5 DADM Administration von Message Queues ... 230
7.6 DGET Lesen einer Nachricht aus einer Service-gesteuerten Queue ... 241
7.7 DPUT Erzeugen zeitgesteuerter Asynchron-Nachrichten ... 251
7.7.1 DPUT-Aufruf ohne Auftrags-Komplex ... 252
7.7.2 DPUT-Aufruf im Auftrags-Komplex ... 265
7.8 FGET Empfangen einer Asynchron-Nachricht ... 272
7.9 FPUT Erzeugen von Asynchron-Nachrichten ... 277
7.10 GTDA Lesen aus einem TLS ... 287
7.11 INFO Informationen abrufen ... 291
7.11.1 INFO CK-Aufruf ... 303
7.12 INIT Initialisieren eines Teilprogramms ... 306
7.13 LPUT Schreiben in die Protokolldatei ... 332
7.14 MCOM Definieren eines Auftrags-Komplexes ... 335
7.15 MGET Empfangen einer Dialog-Nachricht ... 339
7.16 MPUT Senden einer Dialog-Nachricht ... 356
7.17 PADM Administrieren von Druckausgaben und Druckern ... 367
7.18 PEND Beenden eines Teilprogramms ... 375
7.19 PGWT Wartepunkt im Programm setzen ohne Teilprogrammende ... 388
7.20 PTDA Schreiben in einen TLS ... 398
7.21 QCRE Temporäre Queue erzeugen ... 402
7.22 QREL Temporäre Queue löschen ... 406
7.23 RSET Transaktion zurücksetzen ... 409
7.24 SGET Lesen aus einem Sekundärspeicherbereich ... 412
7.25 SIGN An- und Abmelden steuern, Berechtigungsdaten überprüfen, Passwort ändern ... 417
7.25.1 SIGN CL - Locale der Benutzerkennung ändern (BS2000-Systeme) ... 430
7.26 SPUT Schreiben in einen Sekundärspeicherbereich ... 433
7.27 SREL Löschen eines Sekundärspeicherbereichs ... 439
7.28 UNLK Entsperren eines TLS, ULS oder eines GSSB ... 443
8 Programmschnittstelle UTM-HTTP ... 447
8.1 Übersicht über alle UTM-HTTP-Funktionen ... 448
8.1.1 Funktion kcHttpGetHeaderByIndex ... 450
8.1.2 Funktion kcHttpGetHeaderByName ... 452
8.1.3 Funktion kcHttpGetHeaderCount ... 454
8.1.4 Funktion kcHttpGetMethod ... 455
8.1.5 Funktion kcHttpGetMputMsg ... 456
8.1.6 Funktion kcHttpGetPath ... 457

8.1.7 Funktion kcHttpGetQuery 458
8.1.8 Funktion kcHttpGetRc2String 459
8.1.9 Funktion kcHttpGetReqMsgBody 460
8.1.10 Funktion kcHttpGetScheme 461
8.1.11 Funktion kcHttpGetVersion 462
8.1.12 Funktion kcHttpPercentDecode 463
8.1.13 Funktion kcHttpPutHeader 465
8.1.14 Funktion kcHttpPutMgetMsg 467
8.1.15 Funktion kcHttpPutRspMsgBody 468
8.1.16 Funktion kcHttpPutStatus 469
8.2 Adressierung TAC 471
8.3 Bewertung der HTTP-Header eines HTTP-Requests 472
8.4 Versorgung der HTTP-Header einer HTTP-Response 474
8.5 Besonderheiten bei der Kommunikation mit HTTP-Clients 477
8.6 Verhalten in Fehlersituationen 478
9 Event-Funktionen 479
9.1 Event-Exits 481
9.1.1 Event-Exit INPUT 482
9.1.2 Event-Exit HTTP 489
9.1.3 Event-Exit START 493
9.1.4 Event-Exit SHUT 495
9.1.5 Event-Exit VORGANG 496
9.1.6 Event-Exit FORMAT (BS2000-Systeme) 498
9.2 Anwenderspezifische Fehlerbehandlung (Unix- und Linux- Systeme) 506
9.3 STXIT-Routinen (BS2000-Systeme) 508
9.4 Ereignisbehandlung in ILCS-Programmen (BS2000- Systeme) 509
9.5 Event-Services 511
9.5.1 Dialog-Vorgang BADTACS 512
9.5.2 Asynchron-Vorgang MSGTAC 513
9.5.3 Anmelde-Vorgang SIGNON 515
9.5.3.1 Programmierhinweise 516
9.5.3.2 Anmelde-Vorgang für Terminals 517
9.5.3.3 Anmelde-Vorgang für UPIC-Clients und Transportsystem-Anwendungen 520
10 Ergänzungen für C/C++ 523
10.1 Programmaufbau bei C/C++-Teilprogrammen 524
10.1.1 C/C++-Teilprogramme als Unterprogramme 525
10.1.2 Parameter eines C/C++-Teilprogramms 526
10.1.3 Datendeklaration 527
10.1.3.1 Kommunikationsbereich 528
10.1.3.2 Standard Primärer Arbeitsbereich (SPAB) 529
10.1.3.3 Weitere Datenbereiche (AREAs) 530

10.1.4 Datenstrukturen für C/C++-Teilprogramme 534
10.1.5 Befehlsteil eines C/C++-Teilprogramms 536
10.1.6 C/C++-Makroschnittstelle 537
10.1.7 Event-Exits 542
10.1.8 Programmierung der KDCS-Fehlerbehandlung 543
10.1.9 Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme) 544
10.1.9.1 User Signal Routinen (Unix- und Linux-Systeme) 545
10.1.9.2 utmwork-Prozess austauschen (Unix- und Linux-Systeme) 547
10.1.9.3 UTM-Dump erstellen (Unix- und Linux-Systeme) 548
10.1.10 Modifizieren von KDCS-Attributen (BS2000-Systeme) 549
10.1.11 Plattform-spezifische Besonderheiten auf BS2000-Systemen 550
10.1.12 Plattform-spezifische Besonderheiten auf Unix- und Linux-Systemen 552
10.1.13 Plattform-spezifische Besonderheiten auf Windows-Systemen 553
10.2 Programmierbeispiele in C/C++ 554
10.2.1 Beispiele zu einzelnen KDCS-Aufrufen 555
10.2.2 Beispiel für ein vollständiges C-Teilprogramm 559
10.2.3 Beispiel: Event-Exit INPUT (BS2000-Systeme) 560
10.2.4 Beispiel: Event-Service MSGTAC 563
10.2.5 Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen 567
11 Ergänzungen für COBOL 583
11.1 Programmaufbau bei COBOL-Teilprogrammen 584
11.1.1 COBOL-Teilprogramm als Unterprogramm 585
11.1.2 Datenstrukturen für COBOL-Teilprogramme 590
11.1.3 KDCS-Aufrufe in COBOL-Teilprogrammen 593
11.1.4 Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme) 595
11.1.4.1 User Signal Routinen (Unix- und Linux-Systeme) 596
11.1.4.2 utmwork-Prozess austauschen (Unix- und Linux-Systeme) 598
11.1.4.3 UTM-Dump erstellen (Unix- und Linux-Systeme) 599
11.1.5 Plattform-spezifische Besonderheiten auf BS2000-Systemen 600
11.1.6 Plattform-spezifische Besonderheiten auf Unix- und Linux-Systemen 603
11.1.7 Plattform-spezifische Besonderheiten auf Windows-Systemen 606
11.2 Programmier-Beispiele in COBOL 608
11.2.1 Beispiele zu einzelnen KDCS-Aufrufen 609
11.2.2 Beispiel für einen INPUT-Exit (BS2000-Systeme) 616
11.2.3 Beispiel für ein Asynchron-Teilprogramm MSGTAC 618
11.2.4 Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen 622
12 Anhang 634
12.1 Übersicht über alle KDCS-Aufrufe 635
12.2 Unterschiedliche Feldnamen für C/C++ und COBOL 645

12.3 ASCII - EBCDIC Code-Konvertierung 650
12.3.1 BS2000-Systeme 651
12.3.2 Unix-, Linux- und Windows-Systeme 652
12.3.2.1 Code-Konvertierungstabellen auf Unix- und Linux-Systemen modifizieren . 653
12.3.2.2 Code-Konvertierungstabellen auf Windows-Systemen modifizieren 654
13 Fachwörter 655
14 Abkürzungen 696
15 Literatur 701

Anwendungen Programmieren mit KDCS für COBOL, C und C++

1 Einleitung

Die IT-Infrastruktur heutiger Unternehmen als Herzstück und Motor des Geschäftes muss den Anforderungen des digitalen Zeitalters gerecht werden. Dabei muss sie mit vermehrten Datenmengen genauso zurechtkommen wie mit verschärften Anforderungen aus dem Umfeld, z.B. Einhaltung von Compliance-Vorgaben. Ebenso muss die Möglichkeit der kurzfristigen Integration weiterer Applikationen gegeben sein. Und alles dies unter dem Gesichtspunkt einer gewährleisteten Sicherheit.
Somit bestehen wesentliche Anforderungen an eine moderne IT-Infrastruktur u.a. aus

- Flexibilität und schier grenzenloser Skalierbarkeit auch für zukünftige Anforderungen
- hohe Robustheit bei höchster Verfügbarkeit
- absoluter Sicherheit in allen Belangen
- Anpassbarkeit an individuelle Bedürfnisse
- Verursachen geringer Kosten

Fujitsu bietet zur Bewältigung dieser Herausforderungen ein umfangreiches Portfolio innovativer Enterprise Hardware, Software und Support Services im Umfeld unserer Enterprise Mainframe Plattformen an und ist damit Ihr

- verlässlicher Service Provider, der Sie langfristig, flexibel und innovativ beim Betrieb der Mainframe-basierten Kernanwendungen Ihres Geschäftes unterstützt,
- optimaler Partner für die gemeinsame Abdeckung der Anforderungen einer Digitalen Transformation und
- langfristiger Partner aufgrund kontinuierlicher Anpassung moderner Schnittstellen, die eine moderne IT Landschaft mit all ihren Anforderungen mit sich bringt.

Mit openUTM stellt Ihnen Fujitsu eine vielfach erprobte und bewährte Lösung aus dem Middleware-Bereich zur Verfügung.

Die High-End-Plattform für Transaktionsverarbeitung openUTM bietet eine Ablaufumgebung, die all diesen Anforderungen moderner unternehmenskritischer Anwendungen gewachsen ist, denn openUTM verbindet alle Standards und Vorteile von transaktionsorientierten Middleware-Plattformen und Message Queuing Systemen:

- Konsistenz der Daten und der Verarbeitung
- Hohe Verfügbarkeit der Anwendungen
- Hohen Durchsatz auch bei großen Benutzerzahlen, d.h. höchste Skalierbarkeit
- Flexibilität bezüglich Änderungen und Anpassungen des IT-Systems

Eine UTM-Anwendung auf Unix-, Linux- und Windows-Systemen kann auf einem einzelnen Rechner als stand-alone UTM-Anwendung oder auf mehreren Rechnern gleichzeitig als UTM-Cluster-Anwendung betrieben werden.

openUTM ist Teil des umfassenden Angebots von **openSEAS**. Gemeinsam mit der Oracle Fusion Middleware bietet openSEAS die komplette Funktionalität für Anwendungsinnovation und moderne Anwendungsentwicklung. Im Rahmen des Produktangebots **openSEAS** nutzen innovative Produkte die ausgereifte Technologie von openUTM:

- BeanConnect ist ein Adapter gemäß der Java EE Connector Architecture (JCA) und bietet den standardisierten Anschluss von UTM-Anwendungen an Java EE Application Server. Dadurch können bewährte Legacy-Anwendungen in neue Geschäftsprozesse integriert werden.

- Bestehende UTM-Anwendungen können unverändert ins Web übernommen werden. Mit dem UTM-HTTP Interface und dem Produkt WebTransactions stehen in openSEAS zwei Alternativen zur Verfügung, welche es ermöglichen, bewährte Host-Anwendungen flexibel in neuen Geschäftsprozessen und modernen Einsatzszenarien zu nutzen.

Die Produkte BeanConnect und WebTransactions werden im Leistungsüberblick kurz dargestellt. Für diese Produkte gibt es eigene Handbücher.

Wenn im Folgenden von Linux-System bzw. Linux-Plattform die Rede ist, dann ist darunter eine Linux-Distribution wie z.B. SUSE oder Red Hat zu verstehen.

Wenn im Folgenden von Windows-System bzw. Windows-Plattform die Rede ist, dann sind damit alle Windows-Varianten gemeint, auf denen openUTM zum Ablauf kommt.

Wenn im Folgenden von Unix-System bzw. Unix-Plattform die Rede ist, dann ist darunter ein Unix-basiertes Betriebssystem wie z.B. Solaris oder HP-UX zu verstehen.

1.1 Zielgruppe und Konzept des Handbuchs

Das vorliegende Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“ richtet sich an alle, die für die Programmierung von UTM-Anwendungen die Programmschnittstelle KDCS oder die Funktionen der Programmschnittstelle UTM-HTTP nutzen wollen.

Der weitaus größte Teil der Darstellung ist betriebssystem-unabhängig, d.h. die Informationen gelten sowohl für Unix-, Linux- und Windows-Systeme als auch für BS2000-Systeme. Zusätzlich werden Plattform-spezifische Besonderheiten beschrieben. Diese spezifischen Informationen sind durch Randbalken jeweils klar gekennzeichnet. So können Sie - wenn Sie es wünschen - Anwendungen erstellen, die sowohl auf Unix-, Linux- und Windows-Systemen als auch auf BS2000-Systemen einsetzbar sind. Auch wenn Sie bestehende UTM(BS2000) -Anwendungen auf Unix-, Linux- oder auf Windows-Systeme portieren wollen oder umgekehrt, wird Ihnen diese Art der Darstellung die Arbeit erleichtern.

Die Kapitel dieses Handbuchs lassen sich in drei Blöcke gliedern:

- Die Kapitel „Aufbau und Einsatz von UTM-Programmen" bis „Programmaufbau bei Kommunikation mit Transportsystem-Anwendungen" enthalten einführende Informationen. Sie stellen grundlegende Konzepte, wie z. B. den verarbeitungsschritt-modularen Teilprogrammaufbau, die Message Queuing-Funktionalität und die verteilte Verarbeitung, im Zusammenhang dar.
- Kapitel „KDCS-Aufrufe" bis „Event-Funktionen" bilden den Referenzteil, der eher zum Nachschlagen gedacht ist. Kapitel „KDCS-Aufrufe" enthält die KDCS-Aufrufe in alphabetischer Reihenfolge, Kapitel „UTM-HTTP-Aufrufe" die Funktionen der UTM-HTTP-Schnittstelle und Kapitel „Event-Funktionen" die Event-Funktionen.
- Kapitel „Ergänzungen für C/C++" und „Ergänzungen für COBOL" enthalten Sprach-spezifische Informationen, Kapitel „Ergänzungen für C/C++" für die Programmiersprache C/C++, Kapitel „Ergänzungen für COBOL" für COBOL.

Im Anhang finden Sie tabellarische Übersichten, z.B. über die bei den einzelnen KDCS-Aufrufen notwendigen Angaben im Parameterbereich oder über die Rückgaben im Kommunikationsbereich.

Die ausführlichen Verzeichnisse am Ende des Handbuchs - Fachwörter, Abkürzungen, Literatur und Stichwörter - sollen Ihnen den Umgang mit diesem Handbuch erleichtern.

> ¡ Wenn im Folgenden allgemein von Unix-System die Rede ist, dann ist darunter ein Unix-basiertes Betriebssystem wie z.B. Solaris oder HP-UX zu verstehen.
> Wenn im Folgenden allgemein von Linux-System die Rede ist, dann ist darunter eine Linux-Distribution wie z.B. SUSE oder Red Hat zu verstehen.
> Wenn im Folgenden von Windows-System bzw. Windows-Plattform die Rede ist, dann sind damit alle Windows-Varianten gemeint, auf denen openUTM zum Ablauf kommt.

1.2 Wegweiser durch die Dokumentation zu openUTM

In diesem Abschnitt erhalten Sie einen Überblick über die Handbücher zu openUTM und zum Produktumfeld von openUTM.

1.2.1 openUTM-Dokumentation

Die openUTM-Dokumentation besteht aus Handbüchern, den Online-Hilfen für den grafischen Administrationsarbeitsplatz openUTM WinAdmin und das grafische Administrationstool WebAdmin sowie Freigabemitteilungen.

Es gibt Handbücher und Freigabemitteilungen, die für alle Plattformen gültig sind, sowie Handbücher und Freigabemitteilungen, die jeweils für BS2000-Systeme bzw. für Unix-, Linux- und Windows-Systeme gelten.

Sämtliche Handbücher sind im Internet verfügbar unter der Adresse https://bs2manuals.ts.fujitsu.com. Für die Plattform BS2000 finden Sie die Handbücher auch auf der Softbook-DVD.

Die folgenden Abschnitte geben einen Aufgaben-bezogenen Überblick über die Dokumentation zu openUTM V7.0.

Eine vollständige Liste der Dokumentation zu openUTM finden Sie im Literaturverzeichnis.

Einführung und Überblick

Das Handbuch **Konzepte und Funktionen** gibt einen zusammenhängenden Überblick über die wesentlichen Funktionen, Leistungen und Einsatzmöglichkeiten von openUTM. Es enthält alle Informationen, die Sie zum Planen des UTM-Einsatzes und zum Design einer UTM-Anwendung benötigen. Sie erfahren, was openUTM ist, wie man mit openUTM arbeitet und wie openUTM in die BS2000-, Unix-, Linux- und Windows-Plattformen eingebettet ist.

Programmieren

- Zum Erstellen von Server-Anwendungen über die KDCS-Schnittstelle benötigen Sie das Handbuch **Anwendungen programmieren mit KDCS für COBOL, C und C++**, in dem die KDCS-Schnittstelle in der für COBOL, C und C++ gültigen Form und die Programmschnittstelle UTM-HTTP beschrieben sind. Die KDCS-Schnittstelle umfasst sowohl die Basisfunktionen des universellen Transaktionsmonitors als auch die Aufrufe für verteilte Verarbeitung. Es wird auch die Zusammenarbeit mit Datenbanken beschrieben.
  Die Programm-Schnittstelle UTM-HTTP stellt Funktionen zur Verfügung, die für die Kommunikation mit HTTP-Clients verwendet werden können.
- Wollen Sie die X/Open-Schnittstellen nutzen, benötigen Sie das Handbuch **Anwendungen erstellen mit X/Open-Schnittstellen**. Es enthält die openUTM-spezifischen Ergänzungen zu den X/Open-Programmschnittstellen TX, CPI-C und XATMI sowie Hinweise zu Konfiguration und Betrieb von UTM-Anwendungen, die X/Open-Schnittstellen nutzen. Ergänzend dazu benötigen Sie die X/Open-CAE-Spezifikation für die jeweilige X/Open-Schnittstelle.
- Wenn Sie Daten auf Basis von XML austauschen wollen, benötigen Sie das Dokument **XML für openUTM**. Darin werden die C- und COBOL-Aufrufe beschrieben, die zum Bearbeiten von XML-Dokumenten benötigt werden.
- Für BS2000-Systeme gibt es Ergänzungsbände für die Programmiersprachen Assembler, Fortran, Pascal-XT und PL/1.

Konfigurieren

Zur Definition von Konfigurationen steht Ihnen das Handbuch **Anwendungen generieren** zur Verfügung. Darin ist beschrieben, wie Sie mit Hilfe des UTM-Tools KDCDEF sowohl für eine stand-alone UTM-Anwendung als auch für eine UTM-Cluster-Anwendung auf Unix-, Linux- und Windows-Systemen.

- die Konfiguration definieren,
- die KDCFILE erzeugen,
- und im Falle einer UTM-Cluster-Anwendung die UTM-Cluster-Dateien erzeugen.

Zusätzlich wird gezeigt, wie Sie wichtige Verwaltungs- und Benutzerdaten mit Hilfe des Tools KDCUPD in eine neue KDCFILE übertragen, z.B. beim Umstieg auf eine neue Version von openUTM oder nach Änderungen in der Konfiguration. Für eine UTM-Cluster-Anwendung wird außerdem gezeigt, wie Sie diese Daten mit Hilfe des Tools KDCUPD in die neuen UTM-Cluster-Dateien übertragen.

Binden, Starten und Einsetzen

Um UTM-Anwendungen einsetzen zu können, benötigen Sie für das betreffende Betriebssystem (BS2000- bzw. Unix-, Linux- oder Windows-Systeme) das Handbuch **Einsatz von UTM-Anwendungen**.
Dort ist beschrieben, wie man ein UTM-Anwendungsprogramm bindet und startet, wie man sich bei einer UTM-Anwendung an- und abmeldet und wie man Anwendungsprogramme strukturiert und im laufenden Betrieb austauscht. Außerdem enthält es die UTM-Kommandos, die dem Terminal-Benutzer zur Verfügung stehen. Zudem wird ausführlich auf die Punkte eingegangen, die beim Betrieb von UTM-Cluster-Anwendungen zu beachten sind.

Administrieren und Konfiguration dynamisch ändern

- Für das Administrieren von Anwendungen finden Sie die Beschreibung der Programmschnittstelle zur Administration und die UTM-Administrationskommandos im Handbuch **Anwendungen administrieren**. Es informiert über die Erstellung eigener Administrationsprogramme für den Betrieb einer stand-alone UTM-Anwendung oder einer UTM-Cluster-Anwendung sowie über die Möglichkeiten, mehrere UTM-Anwendungen zentral zu administrieren. Darüber hinaus beschreibt es, wie Sie Message Queues und Drucker mit Hilfe der KDCS-Aufrufe DADM und PADM administrieren können.
- Wenn Sie den grafischen Administrationsarbeitsplatz **openUTM WinAdmin** oder die funktional vergleichbare Web-Anwendung **openUTM WebAdmin** einsetzen, dann steht Ihnen folgende Dokumentation zur Verfügung:
  - Die **WinAdmin-Beschreibung** und die **WebAdmin-Beschreibung** bieten einen umfassenden Überblick über den Funktionsumfang und das Handling von WinAdmin/WebAdmin.
  - Das jeweilige **Online-Hilfesystem** beschreibt kontextsensitiv alle Dialogfelder und die zugehörigen Parameter, die die grafische Oberfläche bietet. Außerdem wird dargestellt, wie man WinAdmin bzw. WebAdmin konfiguriert, um stand-alone UTM-Anwendungen und UTM-Cluster-Anwendungen administrieren zu können.

> **i** Details zur Integration von openUTM WebAdmin in den SE Manager des SE Servers finden Sie im SE Server Handbuch **Bedienen und Verwalten**.

Testen und Fehler diagnostizieren

Für die o.g. Aufgaben benötigen Sie außerdem die Handbücher **Meldungen, Test und Diagnose** (jeweils ein Handbuch für Unix-, Linux- und Windows-Systeme und für BS2000-Systeme). Sie beschreiben das Testen einer UTM-Anwendung, den Inhalt und die Auswertung eines UTM-Dumps, das Meldungswesen von openUTM, sowie alle von openUTM ausgegebenen Meldungen und Returncodes.

openUTM-Clients erstellen

Wenn Sie Client-Anwendungen für die Kommunikation mit UTM-Anwendungen erstellen wollen, stehen Ihnen folgende Handbücher zur Verfügung:

- Das Handbuch **openUTM-Client für Trägersystem UPIC** beschreibt Erstellung und Einsatz von Client-Anwendungen, die auf UPIC basieren. Es zeigt auf, was beim Programmieren einer CPI-C-Anwendung zu beachten ist und welche Einschränkungen es gegenüber der Programmschnittstelle X/Open CPI-C gibt.

- Das Handbuch **openUTM-Client für Trägersystem OpenCPIC** beschreibt, wie man OpenCPIC installiert und konfiguriert. Es zeigt auf, was beim Programmieren einer CPI-C-Anwendung zu beachten ist und welche Einschränkungen es gegenüber der Programmschnittstelle X/Open CPI-C gibt.
- Für das mit **BeanConnect** ausgelieferte Produkt **openUTM-JConnect** existiert neben dem Handbuch eine Java-Dokumentation mit der Beschreibung der Java-Klassen.
- Das Handbuch **BizXML2Cobol** beschreibt, wie Sie bestehende Cobol-Programme einer UTM-Anwendung so erweitern können, dass sie als Standard-Web-Service auf XML-Basis genutzt werden können. Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.
- Sie können auch mit dem Software-Produkt WS4UTM (WebServices for openUTM) Services von UTM-Anwendungen als Web Services verfügbar machen. Dazu benötigen Sie das Handbuch **Web-Services für openUTM.** Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.

Kopplung mit der IBM-Welt

Wenn Sie aus Ihrer UTM-Anwendung mit Transaktionssystemen von IBM kommunizieren wollen, benötigen Sie außerdem das Handbuch **Verteilte Transaktionsverarbeitung zwischen openUTM und CICS-, IMS- und LU6.2-Anwendungen**. Es beschreibt die CICS-Kommandos, IMS-Makros und UTM-Aufrufe, die für die Kopplung von UTM-Anwendungen mit CICS- und IMS-Anwendungen benötigt werden. Die Kopplungsmöglichkeiten werden anhand ausführlicher Konfigurations- und Generierungsbeispiele erläutert. Außerdem beschreibt es die Kommunikation über openUTM-LU62, sowie dessen Installation, Generierung und Administration.

Dokumentation zu PCMX

Mit openUTM auf Unix-, Linux- und Windows-Systemen wird die Kommunikationskomponente PCMX ausgeliefert. Die Funktionen von PCMX sind in folgenden Dokumenten beschrieben:

- Handbuch CMX (Unix-Systeme) "Betrieb und Administration" für Unix- und Linux- Systeme
- Online-Hilfe zu PCMX für Windows-Systeme

1.2.2 Dokumentation zum openSEAS-Produktumfeld

Die Verbindung von openUTM zum openSEAS-Produktumfeld wird im openUTM-Handbuch **Konzepte und Funktionen** kurz dargestellt. Die folgenden Abschnitte zeigen, welche der openSEAS-Dokumentationen für openUTM von Bedeutung sind.

Integration von Java EE Application Servern und UTM-Anwendungen

Der Adapter BeanConnect gehört zur Produkt-Suite openSEAS. Der BeanConnect-Adapter realisiert die Verknüpfung zwischen klassischen Transaktionsmonitoren und Java EE Application Servern und ermöglicht damit die effiziente Integration von Legacy-Anwendungen in Java-Anwendungen.

Das Handbuch **BeanConnect** beschreibt das Produkt BeanConnect, das einen JCA 1.5- und JCA 1.6-konformen Adapter bietet, der UTM-Anwendungen mit Anwendungen auf Basis von Java EE, z.B. mit dem Application Server von Oracle, verbindet.

Web-Anbindung und Anwendungsintegration

Anstatt der UTM-HTTP-Programmschnittstelle können Sie alternativ auch das Produkt WebTransactions verwenden. Dann benötigen Sie die Handbücher zu WebTransactions. Die Dokumentation wird durch JavaDocs ergänzt.

1.2.3 Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. den Produkt-spezifischen Readme-Dateien.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter https://bs2manuals.ts.fujitsu.com zur Verfügung. Für die Plattform BS2000 finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen auf BS2000-Systemen

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie auf BS2000-Systemen die folgende Datei:

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

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen. Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemitteilung. Solche Freigabemitteilungen finden Sie online unter https://bs2manuals.ts.fujitsu.com.

1.3 Änderungen in openUTM V7.0

Die folgenden Abschnitte gehen näher auf die Änderungen in den einzelnen Funktionsbereichen ein.

1.3.1 Neue Server-Funktionen

UTM als HTTP-Server

Eine UTM-Anwendung kann auch als HTTP-Server fungieren.

Als Methoden werden GET, PUT, POST und DELETE unterstützt. Neben HTTP wird auch der Zugang über HTTPS unterstützt.

Dazu wurden folgende Schnittstellen geändert:

- Generierung

  Alle Systeme:

  - KDCDEF-Anweisung BCAMAPPL:
    - Beim Operand T-PROT= mit Wert SOCKET gibt es eine zusätzliche Angabe zum Transportprotokoll:

      *USP: Auf Verbindungen dieses Zugriffspunktes soll das UTM-Socket-Protokoll verwendet werden.

      *HTTP: Auf Verbindungen dieses Zugriffspunktes soll das HTTP-Protokoll verwendet werden.

      *ANY: Auf Verbindungen dieses Zugriffspunktes werden sowohl das UTM-Socket-Protokoll als auch das HTTP-Protokoll unterstützt.

    - Beim Operand T-PROT= mit Wert SOCKET gibt es zusätzlich die Angabe zur Verschlüsselung:

      SECURE: Auf Verbindungen dieses Zugriffspunktes erfolgt die Kommunikation unter Verwendung von Transport Layer Security (TLS).

    - Neuer Operand USER-AUTH = *NONE | *BASIC. Hiermit kann angegeben werden, welchen Authentisierungsmechanismus HTTP-Clients für diesen Zugangspunkt verwenden müssen.
  - KDCDEF-Anweisung HTTP-DESCRIPTOR:
    Mit dieser Anweisung wird eine Abbildung des in einem HTTP-Request empfangenen Path auf einen TAC definiert und es können zusätzliche Verarbeitungsparameter angegeben werden.

  BS2000-Systeme:

  - KDCDEF-Anweisung CHAR-SET:
    Mit dieser Anweisung können jeder der von openUTM zur Verfügung gestellten vier Code-Konvertierungen von UTM jeweils bis zu vier Character-Set Namen zugeordnet werden.
- Programmierung
  - KDCS- Kommunikationsbereich (KB):
    Im Kopf des KDCS-Kommunikationsbereichs gibt es im Feld *kccp/KCCP* neue Werte für die Client-Protokolle HTTP, USP-SECURE und HTTPS.
  - KDCS-Aufruf INIT PU:
    - Die Version der Schnittstelle wurde auf 7 erhöht.
    - Um die verfügbare Information vollständig zu erhalten, muss im Feld KCLI der Wert 372 angegeben werden.
    - Neue Felder zur Anforderung (KCHTTP/http_info) und Rückgabe (KCHTTPINF/httpInfo) von HTTP-spezifischen Informationen.
- Administrationsschnittstelle KDCADMI
  - Die Datenstrukturversion von KDCADMI wurde auf Version 11 geändert (Feld *version_data* im Parameterbereich).

- Neue Struktur *kc_http_descriptor_str* im Identifikationsbereich für die Unterstützung des HTTP Deskriptors.
- Neue Struktur *kc_character_set_str* im Identifikationsbereich für die Unterstützung des HTTP Charactersets.
- Neue Felder *secure_soc* und *user_auth* in Struktur *kc_bcamappl_str* für die Unterstützung von HTTP Zugangspunkten.

- Programmschnittstelle UTM-HTTP
  Zusätzlich zum KDCS-Interface bietet UTM ein Interface zum Lesen und Schreiben von HTTP Protokollinformationen und zur Behandlung des HTTP Message Bodys.
  Im Folgenden werden die Funktionen des Interface kurz aufgelistet:

- Funktion *kcHttpGetHeaderByIndex()*
  Diese Funktion liefert den Namen und Wert des HTTP-Header-Feldes für den angegebenen Index zurück.
- Funktion *kcHttpGetHeaderByName()*
  Die Funktion liefert den Wert des über den Namen spezifizierten HTTP-Header-Feldes zurück.
- Funktion *kcHttpGetHeaderCount()*
  Diese Funktion liefert die Anzahl der in dem HTTP-Request enthaltenen Header-Felder zurück, die vom Teilprogramm gelesen werden können .
- Funktion *kcHttpGetMethod()*
  Diese Funktion liefert die HTTP-Methode des HTTP-Requests zurück.
- Funktion *kcHttpGetMputMsg()*
  Diese Funktion liefert die vom Teilprogramm erzeugte MPUT-Nachricht zurück.
- Funktion *kcHttpGetPath()*
  Diese Funktion liefert den mit KC_HTTP_NORM_UNRESERVED normierten HTTP- Path des HTTP-Requests zurück.
- Funktion *kcHttpGetQuery()*
  Diese Funktion liefert die mit KC_HTTP_NORM_UNRESERVED normierte HTTP- Query des HTTP -Requests zurück.
- Funktion *kcHttpGetRc2String()*
  Hilfsfunktion um ein Funktionsergebnis vom Typ enum in einen abdruckbaren null-terminierten String umzuwandeln.
- Funktion *kcHttpGetReqMsgBody()*
  Diese Funktion liefert den Message Body des HTTP Requests zurück.
- Funktion *kcHttpGetScheme()*
  Diese Funktion liefert das Schema des HTTP- Requests zurück.
- Funktion *kcHttpGetVersion()*
  Diese Funktion liefert die Version des HTTP- Requests zurück .
- Funktion *kcHttpPercentDecode()*
  Funktion zur Umwandlung von Zeichen in Prozent-Darstellung in Zeichenfolgen in normale Ein-Zeichen-Darstellung.
- Funktion *kcHttpPutHeader()*
  Diese Funktion übergibt einen HTTP-Header für die HTTP-Response .
- Funktion *kcHttpPutMgetMsg()*
  Diese Funktion übergibt eine Nachricht für das Teilprogramm, die mit MGET gelesen werden kann.
- Funktion *kcHttpPutRspMsgBody()*
  Diese Funktion übergibt eine Nachricht für den Message Body der HTTP-Response.
- Function *kcHttpPutStatus()*
  Diese Funktion übergibt einen HTTP-Statuscode für die HTTP-Response.

- Kommunikation über den Secure Socket Layer (SSL)
  BS2000-Systeme:
  - Ist für eine UTM-Anwendung ein BCAMAPPL mit T-PROT=(SOCKET,...,SECURE) generiert, dann wird beim Anwendungsstart von UTM eine zusätzliche Task mit einem Reverse Proxy gestartet, der für die Anwendung als TLS Termination Proxy fungiert und über den sämtliche SSL-Kommunikation abgewickelt wird.

Unix-, Linux- und Windows-Systeme :

- Für einen sicheren Zugang über TLS steht ein weiterer Netzprozess vom Typ *utmnetssl* zur Verfügung. Sind für eine UTM-Anwendung BCAMAPPL mit T-PROT=(SOCKET,...,SECURE) generiert, dann wird beim Anwendungsstart von UTM eine Anzahl von *utmnetssl* Prozessen gestartet. Die Anzahl dieser Prozesse ist abhängig vom Wert LISTENER-ID dieser BCAMAPPL Objekte. In einem *utmnetssl* Prozess wird für die zugeordneten BCAMAPPL Portnummern die gesamte TLS-Kommunikation abgewickelt.

Verschlüsselung

Die Verschlüsselungsfunktionalität in UTM zwischen einer UTM-Anwendung und einem UPIC-Client wurde überarbeitet. Dabei wurden Sicherheitslücken geschlossen, moderne Methoden aufgenommen und die Auslieferung wie folgt vereinfacht:

- UTM-CRYPT Variante
  Bisher stand die Verschlüsselungsfunktionalität in UTM nur zur Verfügung, wenn man das Produkt UTM-CRYPT installiert hatte. Mit UTM V7.0 ist dies nicht mehr erforderlich. Ab dieser Version wird über die Generierung bzw. zum Anwendungsstart entschieden, ob die Verschlüsselungsfunktionalität zum Einsatz kommt oder nicht.
- Security
  Bei der Kommunikation zwischen einer UTM-Anwendung und einem UPIC-Client wurde eine Sicherheitslücke behoben.

  > **!** Das hat zur Folge, dass verschlüsselte Kommunikation einer UTM-Anwendung V7.0 nur zusammen mit UPIC-Client Anwendungen ab UPIC V7.0 möglich ist!

- Verschlüsselung Level 5 (*Unix-, Linux- und Windows-Systeme):*
  KDCDEF-Anweisungen PTERM, TAC und TPOOL
  Beim Operanden ENCRYPTION-LEVEL gibt es einen zusätzlichen Level 5. Dabei wird zur Vereinbarung des Session-Keys das auf Elliptic Curves basierende Diffie-Hellman Verfahren verwendet und Ein-/Ausgabe-Nachrichten werden mit dem AES-GCM Algorithmus verschlüsselt.

OSI-TP Kommunikation und Portnummern

BS2000-Systeme:

- KDCDEF-Anweisung OSI-CON
  Der Operand LISTENER-PORT kann auch auf BS2000-Systemen angegeben werden.
- Administrationsschnittstelle KDCADMI
  In der Struktur *kc_osi_con_str* wird auch auf BS2000-Systemen im Feld *listener-port* die Portnummer angezeigt.

Subnetze

In einer UTM-Anwendung können auch auf BS2000-Systemen Subnetze generiert werden, um den Zugang zu UTM-Anwendungen auf definierte IP-Adressbereiche beschränken zu können. Zusätzlich kann die Namensauflösung per DNS gesteuert werden.

Dazu wurden folgende Schnittstellen geändert:

- Generierung
  BS2000-Systeme:
  - KDCDEF-Anweisung SUBNET:
    Die SUBNET-Anweisung kann auch auf BS2000-Systemen angegeben werden.

  Alle Systeme:
  - KDCDEF-Anweisung SUBNET:
    Mit RESOLVE-NAMES=YES/NO kann angegeben werden, ob nach einem Verbindungsaufbau eine Namensauflösung per DNS stattfinden soll oder nicht.

    Falls eine Namensauflösung erfolgt, dann wird über die Administrationsschnittstelle und in Meldungen der echte Prozessorname des Kommunikationspartners angezeigt. Andernfalls wird als Prozessorname die IP-Adresse der Komunikationspartners sowie der Name des in der Generierung definierten Subnetzes angezeigt.
- Administrationsschnittstelle KDCADMI
  Die Strukturen *kc_subnet_str* und *kc_tpool_str* enthalten ein neues Feld *resolve_names*.

Zugangsdaten für den XA-Datenbank-Anschluss

Ein modifizierter aber noch nicht aktivierter Benutzername für den XA-Datenbank-Anschluss kann per Administration (KDCADMI) gelesen werden:

- Operationscode KC_GET_OBJECT:
  Datenstruktur *kc_db_info_str*: Neues Feld *db_new_userid*.

Reconnect für den XA-Datenbank-Anschluss

Wird bei einer XA Aktion zur Steuerung der Transaktion entdeckt, dass die Verbindung zur Datenbank nicht mehr besteht, wird versucht die Verbindung zu erneuern und die XA Aktion zu wiederholen.

Nur falls dies nicht erfolgreich ist, werden der betroffene UTM Prozess und die UTM-Anwendung abnormal beendet . Bisher wurde bei jedem Verbindungsverlust zur XA Datenbank unmittelbar ohne erneuten Verbindungs-Versuch die UTM-Anwendung abnormal beendet.

Sonstige Änderungen

- XA-Meldungen
  Die Meldungen bzgl. der XA-Schnittstelle wurden jeweils um die Inserts UTM-Userid und TAC erweitert. Betroffen sind die Meldungen K204-K207, K212-K215 und K217-K218.
- UTM-Tool KDCEVAL
  Im TRACE 2 Satz von KDCEVAL wurde im WAITEND Record der Typ des letzten Auftrags (Börsen-Announcements) aufgenommen (ersten beiden Bytes abdruckbar).

1.3.2 Entfallene Server-Funktionen

Im Einzelnen wurden folgende Funktionen gestrichen:

- Dienstprogramm KDCDEF
  Mehrere Funktionen wurden gestrichen und können nicht mehr in KDCDEF generiert werden. Wenn sie dennoch angegeben werden, wird dies im KDCDEF-Lauf mit einem Syntaxfehler abgelehnt.
  - KDCDEF-Anweisung PTERM
    Operanden-Werte 1 und 2 für ENCRYPTION-LEVEL
  - KDCDEF-Anweisung TPOOL
    Operanden-Werte 1 und 2 für ENCRYPTION-LEVEL
  - KDCDEF-Anweisung TAC
    Operanden-Wert 1 für ENCRYPTION-LEVEL
- *BS2000-Systeme*
  - UTM-Cluster:
    Auf BS2000-Systemen werden UTM-Cluster-Anwendungen nicht mehr unterstützt.
- *Unix-, Linux- und Windows-Systeme*
  - TNS Betrieb:
    Beim Start einer UTM-Anwendung wird die TNS-Generierung nicht mehr gelesen. Die Adressierungsinformation muss vollständig bei der Konfiguration mit KDCDEF hinterlegt werden.

1.3.3 Neue Client-Funktionen

Verschlüsselung

Die Verschlüsselungsfunktionalität in openUTM-Client wurde überarbeitet. Dabei wurden Sicherheitslücken geschlossen, moderne Methoden aufgenommen und die Auslieferung wie folgt vereinfacht:

- UTM-CLIENT-CRYPT Variante
  Bisher stand die Verschlüsselungsfunktionalität in openUTM-Client nur zur Verfügung, wenn man das Produkt UTM-CLIENT-CRYPT installiert hatte. Mit openUTM-Client V7.0 ist dies nicht mehr erforderlich. Ab dieser Version wird zum Ablaufzeitpunkt entschieden ob die Verschlüsselungsfunktionalität zum Einsatz kommt oder nicht.
- Security
  Bei der Kommunikation mit einer UTM-Anwendung wurde eine Sicherheitslücke behoben.
- Verschlüsselung Level 5
  openUTM-Client V7.0 unterstützt die Kommunikation mit UTM V7.0 Anwendungen, bei denen für die Verbindungen zum UPIC-Client ENCRYPTION-LEVEL 5 generiert wurde.
  Bei Level 5 wird zur Vereinbarung des Session-Keys das auf Elliptic Curves basierende Diffie-Hellman Verfahren verwendet und Ein-/Ausgabe-Nachrichten werden mit dem AES-GCM Algorithmus verschlüsselt. AES-GCM unterstützt die Authentifikation und die Verschlüsselung von Nachrichten.

  Der Level 5 wird von openUTM-Client auf allen Plattformen unterstützt.
- Verschlüsselung BS2000
  openUTM-Client (BS2000) verwendet analog zu Unix-, Linux- und Windows-Systemen openSSL anstatt BS2000-CRYPT.

1.3.4 Neue Funktionen für openUTM WinAdmin

WinAdmin unterstützt alle Neuerungen der openUTM V7.0 bzgl. der Programmschnittstelle zur Administration.

1.3.5 Neue Funktionen für openUTM WebAdmin

WebAdmin unterstützt alle Neuerungen der openUTM V7.0 bzgl. der Programmschnittstelle zur Administration.

1.4 Darstellungsmittel

Metasyntax

Die in diesem Handbuch verwendete Metasyntax können Sie der folgenden Tabelle entnehmen:

Formale Darstellung	Erläuterung	Beispiel
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Konstanten (Namen von Aufrufen, Anweisungen, Feldnamen, Kommandos und Operanden etc.), die in dieser Form anzugeben sind.	`LOAD-MODE=STARTUP`
kleinbuchstaben	In Kleinbuchstaben sind in Syntaxdiagrammen und Operandenbeschreibung die Platzhalter für Operandenwerte dargestellt.	`KDCFILE=filebase`
kleinbuchstaben	Im Fließtext werden Variablen sowie Namen von Datenstrukturen und Feldern in kursiven Kleinbuchstaben dargestellt.	*utm-installationsverzeichnis* ist das UTM-Installationsverzeichnis
`Schreibmaschinenschrift`	In Schreibmaschinenschrift werden im Fließtext Kommandos, Dateinamen, Meldungen und Beispiele ausgezeichnet, die in genau dieser Form eingegeben werden müssen bzw. die genau diesen Namen oder diese Form besitzen.	Der Aufruf `tpcall`
{ } und |	In geschweiften Klammern stehen alternative Angaben, von denen Sie eine auswählen müssen. Die zur Verfügung stehenden Alternativen werden jeweils durch einen Strich getrennt aufgelistet.	`STATUS={ ON | OFF }`
[ ]	In eckigen Klammern stehen wahlfreie Angaben, die entfallen können.	`KDCFILE=( filebase [, { SINGLE | DOUBLE } ] )`
( )	Kann für einen Operanden eine Liste von Parametern angegeben werden, sind diese in runde Klammern einzuschließen und durch Kommata zu trennen. Wird nur ein Parameter angegeben, kann auf die Klammern verzichtet werden.	`KEYS=(key1, key2,...key`*n*`)`
Unterstreichen	Unterstreichen kennzeichnet den Standardwert.	`CONNECT= { YES | NO }`
Kurzform	Die Standardkurzform für Anweisungen, Operanden und Operandenwerte wird „fett“ hervorgehoben. Die Kurzform kann alternativ angegeben werden.	`TRANSPORT -SELECTOR =c‘C‘`

Formale Darstellung	Erläuterung	Beispiel

`...`	Punkte zeigen die Wiederholbarkeit einer syntaktischen Einheit an.
	Außerdem kennzeichnen die Punkte Ausschnitte aus einem Programm, einer Syntaxbeschreibung o.ä.
		`KDCDEF starten`
		`...`
		`OPTION`
		`DATA=statement_file`
		`...`
		`END`

Symbole

für Verweise auf umfassende und detaillierte Informationen zum jeweiligen Thema.

 für Hinweistexte.

 für Warnhinweise.

Sonstiges

utmpfad bezeichnet auf Unix-, Linux- und Windows-Systemen das Verzeichnis, unter dem openUTM installiert wurde.

filebase bezeichnet auf Unix-, Linux- und Windows-Systemen das Dateiverzeichnis der UTM-Anwendung. Dies ist der Basisname, der in der KDCDEF-Anweisung MAX KDCFILE= generiert wurde.

$userid bezeichnet auf BS2000-Systemen die Kennung, unter der openUTM installiert wurde.

upic-dir bezeichnet auf Unix-, Linux- und Windows-Systemen das Verzeichnis, unter dem openUTM-Client für Trägersystem UPIC installiert ist.

2 Aufbau und Einsatz von UTM-Programmen

In diesem Kapitel erhalten Sie einen ersten Überblick über die Programmierung von UTM-Anwendungen. Sie erfahren, was Vorgänge und Teilprogramme sind und wie Sie die Konzepte von openUTM einsetzen können, um die Geschäftslogik Ihrer Anwendung zu realisieren.

Eine UTM-Anwendung stellt ihren Benutzern bestimmte Services zur Verfügung: Sie erledigt Service-Requests (Aufträge), die von Terminal-Benutzern, Client-Programmen oder von anderen Anwendungen an sie gestellt werden.

Ein **Vorgang**, der auch **Service** genannt wird, dient der Bearbeitung eines Auftrags an eine UTM-Anwendung. Er setzt sich aus einer oder mehreren Transaktionen und einem oder mehreren Programmläufen zusammen. openUTM unterscheidet dabei zwischen Dialog-Vorgängen und Asynchron-Vorgängen. Ein UTM-Vorgang entspricht im allgemeinen einem Geschäftsvorfall der Anwendungslogik.

Bei der Erstellung einer Anwendung programmieren Sie die Geschäftslogik Ihrer Anwendung in der Form von **Teilprogrammen**, die auch **Service-Routinen** genannt werden. Die Teilprogramme laufen als Unterprogramme und unter Kontrolle der Main Routine einer UTM-Anwendung; die Main Routine ist Bestandteil des Systemcodes von openUTM.

Mit der in den Teilprogrammen realisierten Geschäftslogik legen Sie die Leistung der Services fest, die Sie den Benutzern Ihrer Anwendung zur Verfügung stellen wollen. Die Teilprogramme können Sie in einer der gängigen Programmiersprachen (C, C++, COBOL, u.a.) erstellen.

Aus den Teilprogrammen können Sie auf UTM-Systemfunktionen und externe Resource Manager wie z.B. Datenbanken zugreifen. Die Teilprogramme nutzen die UTM-Systemfunktionen durch integrierte **UTM-Aufrufe**, z.B. zur Transaktionssicherung oder zum Senden von Nachrichten an Kommunikationspartner.

Für diese UTM-Aufrufe können Sie verschiedene Schnittstellen verwenden: neben der in diesem Handbuch beschriebenen KDCS-Schnittstelle stehen Ihnen auch die X/Open-Schnittstellen CPI-C, XATMI und TX zur Verfügung (siehe openUTM-Handbuch „Anwendungen erstellen mit X/Open-Schnittstellen“).

Zusätzlich stellt Ihnen openUTM mit der Programm-Schnittstelle UTM-HTTP noch Funktionen zur Verfügung, die Sie bei der Kommunikation mit HTTP-Clients verwenden können. Diese sind im Kapitel „UTM-HTTP-Aufrufe" beschrieben.

Charakteristika der KDCS-Schnittstelle

Die Schnittstelle KDCS (Kompatibles Datenkommunikationssystem) wurde als herstellerneutrale Schnittstelle für transaktionsorientierte Anwendungen definiert und genormt (DIN 66 265). openUTM unterstützt den vollen Umfang dieser Norm und bietet darüber hinaus wesentliche Erweiterungen, z.B. für die verteilte Verarbeitung. Einen Überblick über die Erweiterungen gibt die Tabelle in Kapitel "Übersicht über alle KDCS-Aufrufe".

KDCS zeichnet sich durch folgende Charakteristika aus:

- umfassende Palette von Funktionsaufrufen für universalen Einsatz
  (z.B. auch für Pseudo Conversations, Message Queuing oder die unmittelbare Kommunikation mit Terminals)
- KDCS-spezifische Speicherbereiche für einfache und sichere Programmierung
- Event-Funktionen für Ereignissteuerung

KDCS steht für die Programmiersprachen C, C++ und COBOL zur Verfügung; auf BS2000-Systemen zusätzlich für Assembler, Fortran, PL/I und Pascal-XT.

UTM-Anwendungsprogramm - UTM-Anwendung

Ein UTM-Anwendungsprogramm besteht aus der UTM-Main Routine KDCROOT und den UTM-Teilprogrammen.

Die Main Routine KDCROOT steuert als Teil des UTM-Systemcodes den Ablauf der Anwendung. Sie wird beim Generieren der Anwendung erzeugt
(siehe openUTM-Handbuch „Anwendungen generieren“).

Damit die UTM-Teilprogramme unter dem Management von openUTM ablaufen können, binden Sie die übersetzten Service-Routinen, zusammen mit weiteren Modulen (Zuordnungstabellen, Meldungen, benutzte Bibliotheken etc.) und der Main Routine KDCROOT zum **UTM-Anwendungsprogramm** (siehe openUTM-Handbuch „Einsatz von UTM-Anwendungen auf BS2000-Systemen“ und openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“).
Das Binden kann statisch (d.h. vor dem Start der Anwendung) oder dynamisch (d.h. beim Start oder während des Betriebs der Anwendung) erfolgen.

Beim Start der **UTM-Anwendung** wird das UTM-Anwendungsprogramm in einer von Ihnen festgelegten Anzahl von Prozessen gestartet. Technisch gesehen ist eine UTM-Anwendung also eine Prozessgruppe, die zur Laufzeit eine logische Server-Einheit bildet.

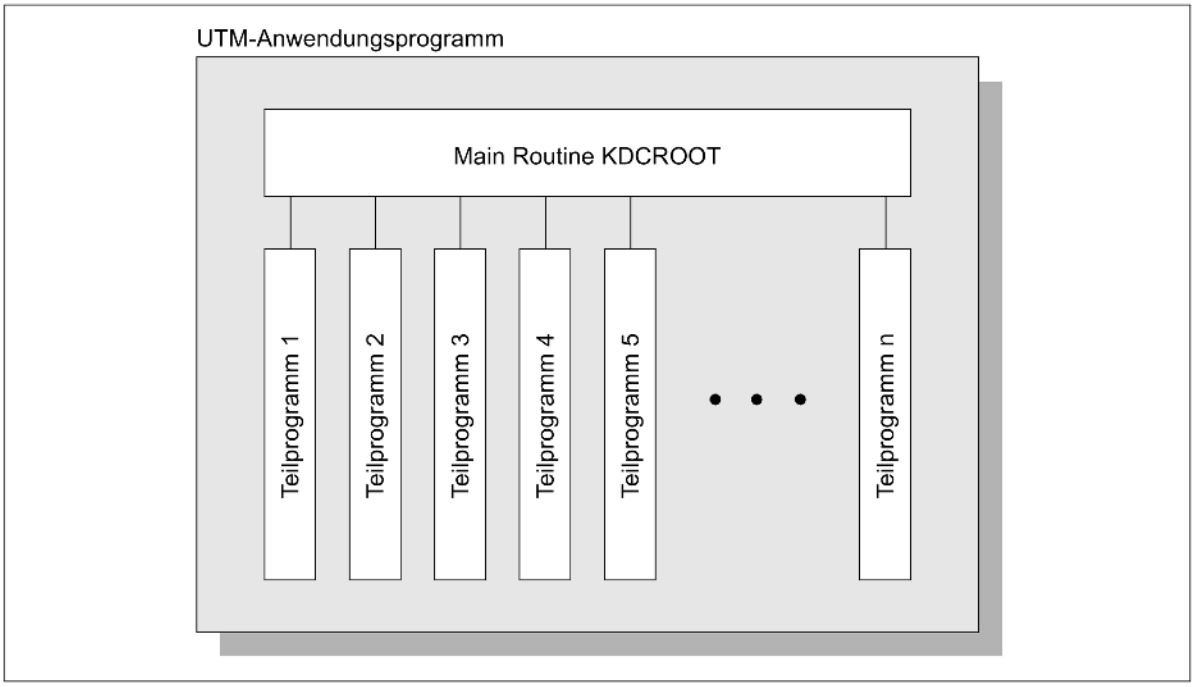


Bild: Main Routine KDCROOT und mehrere Teilprogramme

Die Zusammenarbeit zwischen den Teilprogrammen und der Main Routine KDCROOT ist über **KDCS-Aufrufe** realisiert. Mit KDCS-Aufrufen in einem Teilprogramm teilen Sie der Main Routine KDCROOT mit, welche Funktion openUTM ausführen soll. Die hierfür notwendigen Angaben machen Sie im KDCS-Parameterbereich, dessen Adresse Sie bei jedem KDCS-Aufruf als ersten Parameter übergeben.

Für die Strukturierung des KDCS-Parameterbereichs stehen Ihnen vordefinierte Sprachspezifische Datenstrukturen zur Verfügung, für COBOL im COPY-Element KCPAC, für C/C++ in der Include-Datei *kcmac.h*. Wie Sie diesen Bereich für die einzelnen KDCS-Aufrufe jeweils versorgen müssen, ist in sprachunabhängiger Form im Kapitel „KDCS-Aufrufe“ beschrieben. Die Sprach-spezifischen Besonderheiten finden Sie in den Kapiteln „Ergänzungen für C/C++“ und „Ergänzungen für COBOL“.

In den folgenden Abschnitten erfahren Sie, welche Möglichkeiten es gibt, ein Anwendungsprogramm zu strukturieren.

> **i** Wenn in diesem Kapitel von "Programm" oder "Teilprogramm" die Rede ist, so ist damit der Ablauf dieses Teilprogramms gemeint und nicht der Programmtext. Wenn z.B. von der "Reihenfolge der Aufrufe in einem Teilprogramm" gesprochen wird, so ist dies die Reihenfolge, in der diese Aufrufe durchlaufen werden und nicht die Reihenfolge, in der diese Aufrufe im Quellprogramm stehen. Dieser Ablauf wird auch **Teilprogrammlauf** genannt.

2.1 Vorgangs-Konzept von openUTM

Start eines Vorgangs

Jedem Teilprogramm werden bei der Generierung der Anwendung oder mittels dynamischer Konfiguration ein oder mehrere Transaktionscodes zugeordnet.

Der Transaktionscode des ersten Teilprogramms eines Vorgangs hat dabei eine besondere Funktion, da über diesen der Vorgang gestartet wird. Dieser Transaktionscode wird auch Vorgangs-Transaktionscode oder kurz Vorgangs-TAC genannt. Für jeden gestarteten Vorgang richtet openUTM einen spezifischen Kontext ein (Speicherbereiche etc.).

Der Vorgangs-TAC kann auf die verschiedensten Arten eingegeben werden, z.B. unmittelbares Eintippen am Terminal, Wahl eines Menüpunktes in einem alphanumerischen Format, Mausklick an einem GUI-Client oder auch über einen Web-Browser.

Zusammen mit dem Transaktionscode kann eine Nachricht an openUTM übergeben werden, die die für die gewünschte Verarbeitung notwendigen Daten enthält.

> ¡ Für den Begriff "Vorgang" wird oftmals auch der allgemeinere Begriff "Service" verwendet.

Dialog-Schritt und Verarbeitungsschritt

Im einfachsten Fall umfasst ein Vorgang nur einen Dialog-Schritt, d.h. zur Bearbeitung der Anforderung (Service-Request) sind keine weiteren Interaktionen notwendig: Auf die Eingabe des Vorgangs-TACs folgt die Ausgabe des Ergebnisses, z.B. "Alles erledigt".

In vielen Fällen wird dieses Schema jedoch nicht ausreichen. Zusätzliche Daten müssen angefordert werden, Zwischenergebnisse angezeigt, individuelle Wahlmöglichkeiten und Verzweigungen im Vorgangs-Ablauf vorgesehen werden: Ein Vorgang umfasst also oft mehrere Dialog-Schritte.

Ein **Dialog-Schritt** beginnt mit einer Dialog-Nachricht, die ein Kommunikationspartner an die UTM-Anwendung sendet, und endet mit der Dialog-Nachricht, die der Vorgang demselben Kommunikationspartner als Antwort sendet. Zwischen diesen beiden Zeitpunkten werden die Daten verarbeitet, es findet keine Kommunikation mit diesem Kommunikationspartner statt.

Bei verteilter Verarbeitung kommuniziert ein Vorgang nicht nur mit dem Benutzer, der den Vorgang gestartet hat, sondern auch mit einem oder mehreren Partner-Vorgängen: Ein vom Benutzer gestarteter Vorgang sendet also die nächste Nachricht möglicherweise nicht an den Benutzer, sondern an eine andere Server-Anwendung. Da es sich um keine Antwort handelt, spricht man in diesem Fall nicht von einem Dialog-Schritt, sondern von einem Verarbeitungsschritt: Ein **Verarbeitungsschritt** beginnt mit dem Empfangen einer Dialog-Nachricht und endet mit dem Senden der nächsten Dialog-Nachricht. Diese kann eine Antwort an denselben Partner sein (dann entspricht der Verarbeitungsschritt einem Dialog-Schritt), oder auch eine Nachricht an einen Dritten.

Ein Vorgang kann also in mehrere Dialog-Schritte gegliedert sein, ein Dialog-Schritt - bei verteilter Verarbeitung - in mehrere Verarbeitungsschritte.

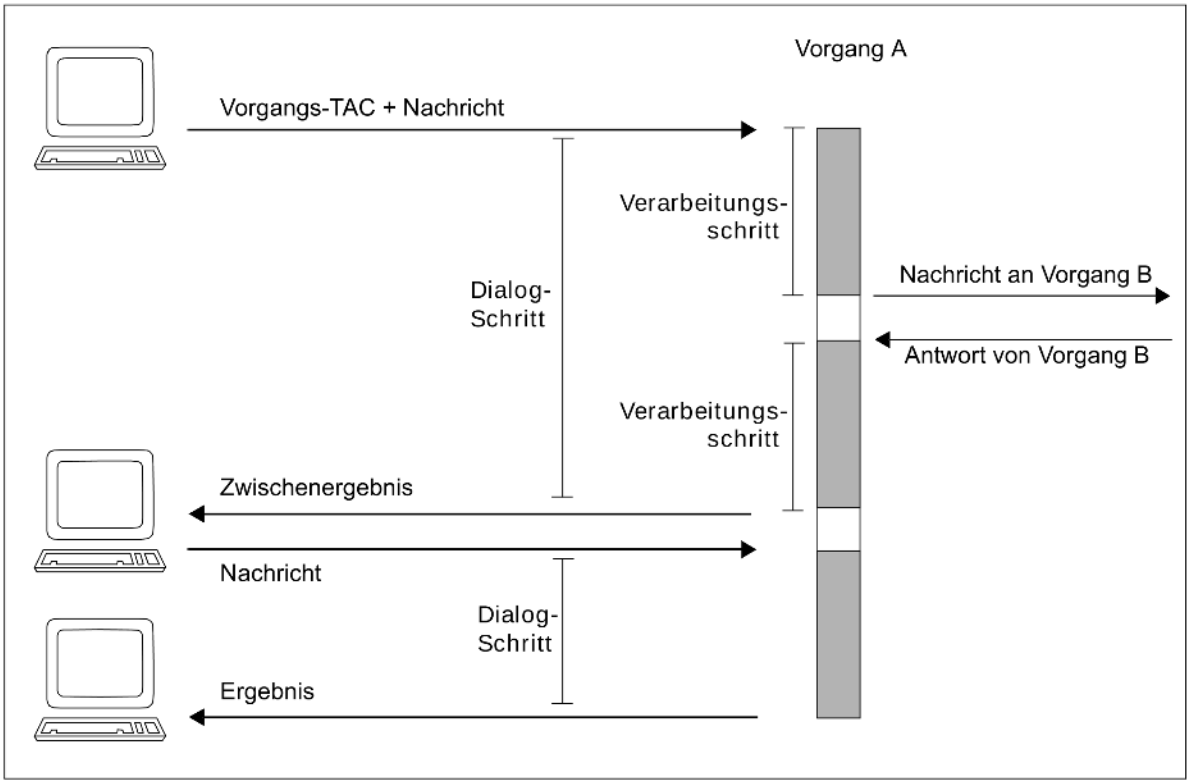


Bild: Dialog-Schritte und Verarbeitungsschritte

Abwechseln von Eingaben und Antworten

Für den Aufbau von Dialog-Vorgängen fordert openUTM einen "strengen Dialog", d.h. auf jede Nachricht muss eine Antwort folgen. Nach dem Senden einer Nachricht an einen Vorgang muss also erst eine Antwort empfangen werden, bevor wieder eine Nachricht an diesen Partner gesendet werden kann.

Dieses Schema - zusammen mit dem verarbeitungsschritt-modularen Vorgangs-Aufbau - ermöglicht es openUTM, die Prozesse optimal zu nutzen (siehe die nächsten beiden Abschnitte).

Verarbeitungsschritt-modularer Vorgangs-Aufbau

Wenn ein Vorgang mehrere Dialog- oder Verarbeitungsschritte umfasst, besteht der Vorgang in der Regel nicht aus **einer** Service-Routine, sondern setzt sich aus einer Folge separater Service-Routinen zusammen, die **Teilprogramme** genannt werden. Im Standardfall entspricht dabei ein Teilprogramm einem Verarbeitungsschritt, d. h. ein Teilprogramm liest eine Nachricht und gibt am Ende eine Nachricht aus. Der Prozess wird automatisch sofort freigegeben und steht für andere Aufträge bereit. Das nächste Teilprogramm startet erst, wenn die nächste Nachricht vom Kommunikationspartner eintrifft.

Während z.B. der Benutzer am Terminal die Ausgabe liest und die nächste Eingabe vorbereitet, wird vom Vorgang kein Prozess belegt. Wenn der Terminal-Benutzer dann seine Eingabe abgeschlossen hat, übernimmt - ohne dass der Benutzer oder das Teilprogramm dies bemerken - u.U. ein anderer Prozess die Fortsetzung des Dialogs.

openUTM sorgt so für eine optimale Auslastung der Prozesse, was sich positiv auf die Performance auswirkt. Dieses Dialog-Konzept, auch "**pseudo-conversational**" genannt, setzt openUTM nicht nur beim Dialog mit Terminal-Benutzern ein, sondern auch bei der Programm-Programm-Kommunikation.

Das verarbeitungsschritt-modulare Grundschema vereinfacht zudem das Design von Anwendungen und sorgt für übersichtliche Programme. Es engt den Programmierer jedoch nicht ein, da es flexibel eingesetzt werden kann und openUTM diverse Variationsmöglichkeiten bietet.

Beispiele für Mehrschritt-Vorgänge und nähere Informationen zur Teilprogramm-Verknüpfung finden Sie im Abschnitt „Strukturierung von Vorgängen“.

Asynchron-Vorgänge

Mit openUTM können Sie nicht nur Vorgänge definieren, die im Dialog mit dem Benutzer ablaufen, sondern auch Vorgänge, die entkoppelt vom Benutzer gestartet werden. Die in openUTM integrierte Message Queuing-Funktionalität ermöglicht es z.B. besonders umfangreiche oder zeitunkritische Aufgaben - wie langlaufende Statistikberechnungen oder Sortierarbeiten - von Online-Dialogen zu entkoppeln, ohne dass auf Transaktionssicherheit verzichtet werden müsste. Die Message Queuing-Funktionalität steht nicht nur für Verarbeitungsaufgaben, sondern auch für die Ausgabe von Nachrichten zur Verfügung, z.B. für Druckaufträge, Nachrichten an ein Terminal oder Nachrichten an Service-gesteuerte Queues ("Nachrichten an Service-gesteuerte Queues").

Das Message Queuing-Konzept und die Einsatzmöglichkeiten werden im openUTM-Handbuch „Konzepte und Funktionen“ vorgestellt, weitere Informationen finden Sie im vorliegenden Handbuch in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

2.2 Aufbau eines Teilprogramms

Bevor der Aufbau komplexerer Vorgänge beschrieben wird, soll in diesem Abschnitt der Aufbau eines einzelnen Teilprogramms dargestellt werden. Zunächst wird auf den Programmrahmen eingegangen, der für alle Arten von Teilprogrammen gilt, danach auf den Aufbau eines Dialog-Teilprogramms.

Der Aufbau eines Asynchron-Teilprogramms wird im Abschnitt „Aufbau eines Asynchron-Vorgangs“ behandelt.

2.2.1 Programmrahmen

Die Zusammenarbeit zwischen den Teilprogrammen und openUTM ist über **KDCS-Aufrufe** realisiert. Mit diesen Aufrufen teilen Sie openUTM mit, welche Funktion ausgeführt werden soll. Bei jedem KDCS-Aufruf übergeben Sie die Adresse des **KDCS-Parameterbereichs**, sowie bei manchen Aufrufen die Adresse eines Nachrichtenbereichs.

Im KDCS-Parameterbereich werden die Aufrufparameter an openUTM übergeben. Für den KDCS-Parameterbereich stehen Ihnen vordefinierte Sprach-spezifische Datenstrukturen zur Verfügung, für COBOL im COPY-Element KCPAC, für C/C++ in der Include-Datei *kcpa.h*. Wie Sie diesen Bereich für die einzelnen KDCS-Aufrufe versorgen müssen, ist in sprachunabhängiger Form im Kapitel „KDCS-Aufrufe“ beschrieben.

Rückgabeinformationen übergibt openUTM nach jedem KDCS-Aufruf (außer PEND) im **KB-Rückgabebereich.** Die Auswertung insbesondere des Returncodes gibt Aufschluss über die erfolgreiche oder gescheiterte Ausführung und kann zu entsprechenden Steuerungsmaßnahmen im Programm benutzt werden (siehe auch Abschnitt „Programmierung von Fehlerroutinen“).

Zusätzlich finden Sie nach jedem KDCS-Aufruf im **KB-Kopf** jeweils aktuelle Informationen über Benutzer, Vorgang, Teilprogramm und Kommunikationspartner.

KB-Kopf und KB-Rückgabebereich sind Teil des **Kommunikationsbereichs (KB)**, siehe "Kommunikationsbereich (KB)". Die Adresse des KB stellt openUTM einem Teilprogramm beim Aufruf über einen Aufrufparameter zur Verfügung.

Für die Struktur der Bereiche KB-Kopf und KB-Rückgabebereich stehen Ihnen Sprachspezifische Datenstrukturen zur Verfügung - für COBOL im COPY-Element KCKBC, für C/C++ in der Include-Datei *kcca.h*.

Der erste UTM-Aufruf in einem Teilprogrammlauf ist der INIT-Aufruf. Mit diesem Aufruf leitet ein Teilprogramm die Zusammenarbeit mit openUTM ein. Vor dem INIT-Aufruf darf anderer Programmcode stehen. Nach dem INIT-Aufruf stellt openUTM dem Teilprogramm aktuelle, Ablauf-spezifische Informationen im KB-Kopf und im Nachrichtenbereich zur Verfügung.

Der letzte Aufruf in einem Teilprogrammlauf ist der PEND-Aufruf. Mit diesem Aufruf wird das Teilprogramm beendet; das Teilprogramm erhält danach die Kontrolle nicht wieder zurück.

Mit den verschiedenen Varianten des PEND-Aufrufs steuern Sie den Ablauf eines UTM-Vorgangs. Dazu haben Sie die folgenden Möglichkeiten:

PEND PR	setzt den Verarbeitungsschritt in einem anderen Teilprogramm fort ohne dass ein Nachrichtenaustausch mit Partner erfolgt.
PEND PA	wirkt wie PEND PR.
PEND PS	speziell für den Anmelde-Vorgang, ähnlich wie PEND PR.
PEND KP	beendet den Verarbeitungsschritt, aber nicht die Transaktion.
PEND SP	beendet die Transaktion, aber nicht den Verarbeitungsschritt.
PEND RE	beendet den Verarbeitungsschritt und gleichzeitig die Transaktion.
PEND FI	beendet Verarbeitungsschritt, Transaktion und Vorgang.
PEND FC	beendet Transaktion und Vorgang und setzt den Verarbeitungsschritt in einem anderen Vorgang fort.
PEND RS	bricht den Verarbeitungsschritt ab und setzt die Transaktion zurück auf den letzten Sicherungspunkt.

PEND ER bricht den Verarbeitungsschritt ab, setzt die Transaktion zurück, beendet den Vorgang und erzeugt einen UTM-Dump. Auf BS2000-Systemen wird das Anwendungsprogramm nachgeladen, auf Unix-, Linux- und Windows-Systemen wird es neu gestartet.

PEND FR wirkt wie PEND ER, ohne Nachladen bzw. Neustart des Anwendungsprogramms.

Beachten Sie, dass viele Aktionen - wie z.B. das Senden der Ausgabe-Nachrichten an die Kommunikationspartner eines UTM-Vorgangs - von openUTM erst beim PEND-Aufruf ausgeführt werden und nicht schon während des Teilprogrammlaufs. Dieses Verhalten hat seine Ursache in der transaktionsorientierten Arbeitsweise von openUTM; bis zum Ende einer Transaktion können Sie entscheiden, ob die von einem Teilprogramm in dieser Transaktion durchgeführten Aktionen gültig gesetzt, oder aber rückgängig gemacht werden sollen.

Tritt in einer UTM-Transaktion ein schwerer Fehler auf, setzt openUTM selbstständig die ganze Transaktion auf den letzten Sicherungspunkt zurück und beendet den Vorgang (siehe auch openUTM-Handbuch „Konzepte und Funktionen“).

2.2.2 Aufbau eines Dialog-Teilprogramms

Für Dialog-Teilprogramme fordert openUTM einen "strengen Dialog", d.h. auf jede Nachrichteneingabe muss eine Nachrichtenausgabe folgen, die entweder das Ergebnis oder eine Fehlermeldung liefert.

Nach dem **INIT** kann das Teilprogramm mit **MGET** die Dialog-Nachricht lesen, die von einem Terminal, von einem Client-Programm oder von einer anderen Anwendung stammen kann.

Diese Nachricht kann sein:

- eine vollständige Nachricht
- eine Teilnachricht
- eine leere Nachricht, wenn z.B. nur ein TAC eingegeben wurde
- eine Rücksetznachricht von einem Teilprogramm, das mit PEND RS beendet wurde
- ein Returncode, wenn z.B. der Benutzer eine Funktions-Taste betätigt hat. In diesem Fall muss ggf. die Nachricht mit einem weiteren MGET gelesen werden.
- bei verteilter Verarbeitung eine Statusinformation, z.B. von einem Auftragnehmer, der sich wegen eines Fehlers beendet hat
- bei verteilter Verarbeitung über OSI TP eine Handshake-Aufforderung (vom Partner mit MPUT HM gesendet) oder eine negative Handshake-Quittung (mit MPUT EM gesendet)

Nach der Verarbeitung der Eingabe müssen Sie entweder den Verarbeitungsschritt in einem anderen Teilprogramm fortsetzen oder mit **MPUT** die Antwort auf die Anfrage des Partners ausgeben (Dialog-Schrittende) - bei verteilter Verarbeitung kann die Nachricht auch an einen Auftragnehmer-Vorgang gerichtet werden (kein Ende des Dialog-Schritts sondern lediglich Verarbeitungsschritt-Ende).

Letzter UTM-Aufruf in Ihrem Teilprogramm muss ein **PEND** sein, wie im Abschnitt „Programmrahmen“ beschrieben.

Wenn der PEND-Aufruf den Verarbeitungsschritt abschließt, gibt openUTM die Nachricht nach der PEND-Bearbeitung an das Terminal, das Client-Programm oder an die andere Anwendung aus.

> ¡ Im Normalfall muss vor jedem PEND-Aufruf, der einen Verarbeitungsschritt abschließt, ein MPUT-Aufruf abgesetzt werden. Bei Ausnahmen wird jeweils gesondert darauf hingewiesen.

Das folgende Bild zeigt den Grundaufbau eines Dialog-Teilprogramms.

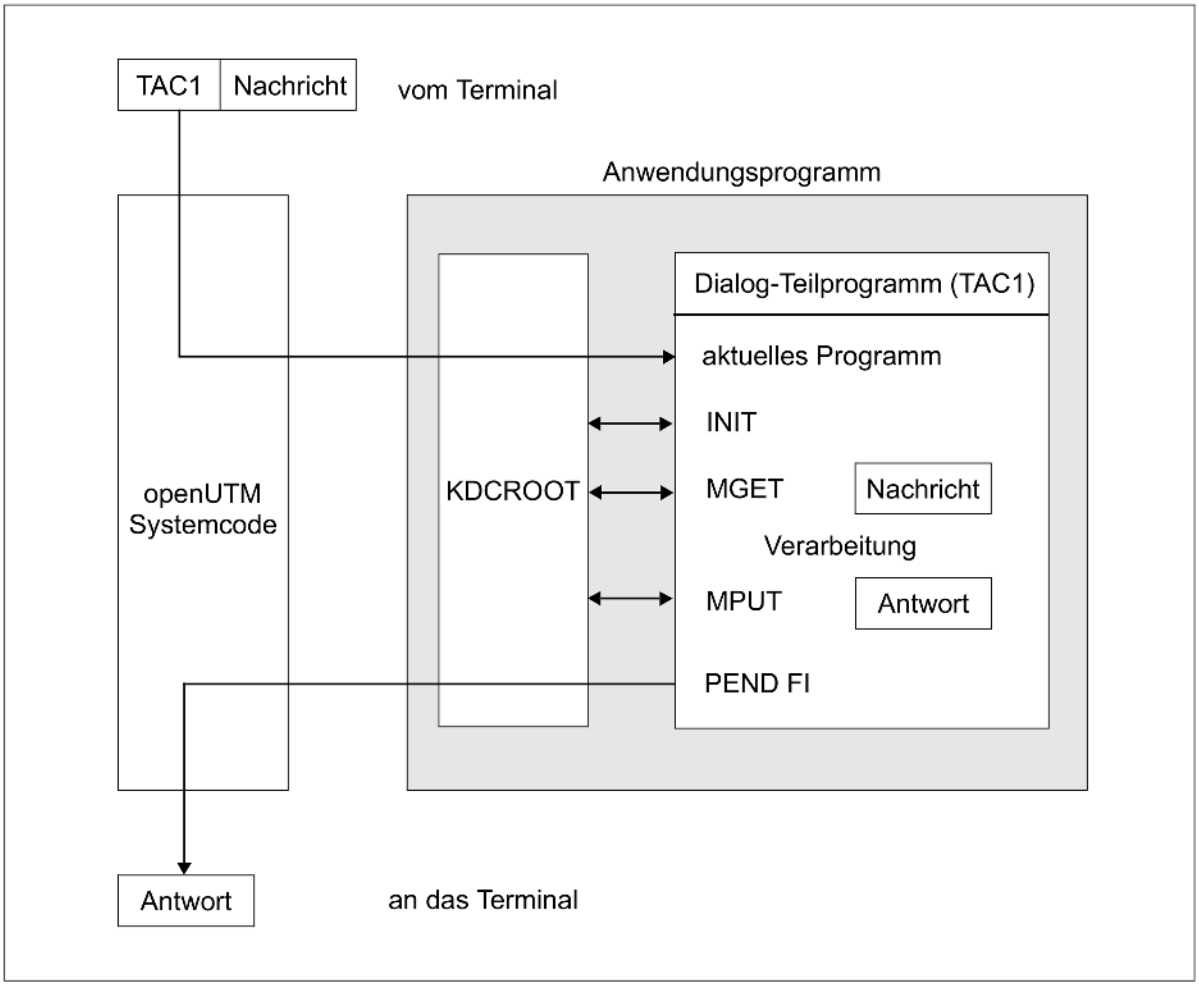


Bild: Aufbau eines Dialog-Programms

Der Transaktionscode "TAC1" wird vom Terminal-Benutzer eingegeben. TAC1 wurde bei der Generierung dem Teilprogramm zugeordnet (KDCDEF-Anweisung TAC, Operand PROGRAM=aktueller Programmname).

2.2.3 Reentrant-Fähigkeit von Teilprogrammen

Ein Teilprogramm wird bei openUTM immer durch einen Transaktionscode aufgerufen. Das Teilprogramm wird jedoch nicht bei jedem Aufruf neu geladen, sondern eine Programmkopie im virtuellen Speicher bearbeitet nacheinander unabhängige Aufträge. Konkret heißt dies, dass nach einem PEND ein Prozesswechsel stattfinden kann, so dass das Folgeteilprogramm in einem anderen Prozess läuft und damit ggf. auch eine andere Datenumgebung vorfindet.

Daher muss ein UTM-Teilprogramm seriell wiederverwendbar, d.h. reentrant-fähig sein:

- Programm-spezifische Daten müssen zu Beginn des Teilprogrammlaufs in den Ausgangszustand versetzt werden.
- Ein Programm-spezifisches Datenfeld darf erst gelesen werden, wenn es vorher im gleichen Teilprogrammlauf beschrieben wurde.

openUTM erleichtert Ihnen die Programmierung reentrant-fähiger Programme durch einen speziellen, von openUTM verwalteten Teilprogramm-spezifischen Speicherbereich (SPAB, siehe "Standard Primärer Arbeitsbereich (SPAB)"). Wenn Sie für alle variablen Daten diesen Speicherbereich verwenden, wird die Reentrant-Fähigkeit von openUTM automatisch gewährleistet.

Dieses heißt konkret für COBOL-Teilprogramme, dass Variablen in der WORKING-STORAGE SECTION beim Aufruf des Teilprogramms wieder initialisiert werden müssen, so weit sie in einem vorherigen Teilprogrammlauf beschrieben worden sind. Für C/C++ gilt die letzte Aussage für die Variablen mit dem Speicherklassenattribut *static* bzw. *extern* und den Variablen mit einer externen Bindung (modulglobale Variablen ohne Speicherklassenattribut). Für Variablen mit prozessübergreifendem Geltungsbereich (liegen im Shared Memory) gilt zusätzlich, dass zumindest ein schreibender Zugriff hierauf serialisiert werden muss. Die oben genannten Variablen können aber ohne Einschränkung zum Nur-Lesen benutzt werden.

Für C/C++ dürfen unbedenklich Variablen mit dem Speicherklassenattribut *auto* bzw. *register* verwendet werden; diese müssen dann aber vor dem Lesen beschrieben werden.

2.3 Strukturierung von Vorgängen

Ein Vorgang kann aus einem oder mehreren Teilprogrammen bestehen.
Wie ein Vorgang aufgebaut ist, der nur aus einem Teilprogramm besteht und nur einen Verarbeitungsschritt bearbeitet (Einschritt-Vorgang), zeigte bereits das Bild auf der "Aufbau eines Dialog-Teilprogramms":

Ein einzelnes Teilprogramm bearbeitet die gestellte Aufgabe vollständig in einem Schritt und beendet sich nach der Verarbeitung mit PEND FI.

Für komplexe Aufgaben, deren Lösung mehrere Schritte benötigen, können Sie einen Vorgang oder eine Transaktion in mehrere Teile strukturieren. Dafür stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Mehrschritt-Vorgänge
- mehrere Teilprogramme in einem Verarbeitungsschritt
- mehrere Verarbeitungsschritte in einem Teilprogramm
- Unterprogramm-Aufrufe aus Teilprogrammen
- Vorgänge ketten
- Vorgänge kellern

Diese Möglichkeiten können Sie auch miteinander kombinieren.

Bei der Konstruktion Ihrer UTM-Teilprogramme haben Sie freie Hand: Bindend sind lediglich der Programmrahmen mit INIT und PEND sowie bei Verarbeitungsschritten der MPUT.

2.3.1 Mehrschritt-Vorgänge

Der Dialog in mehreren Verarbeitungsschritten ist eine häufige Art der Transaktionsverarbeitung. Den Benutzern der Anwendungen soll mit diesem Verfahren die Arbeit erleichtert werden. Sie sollen den Auftrag schrittweise interaktiv formulieren können, damit sie die Teilergebnisse bei jedem Verarbeitungsschritt verwerten können.

Die Platzbuchung eines Zuges lässt sich z.B. als Zweischritt-Vorgang programmieren:

1. Schritt: Freie Plätze eines Zugs abfragen.

2. Schritt: Platz belegen und Reservierung bestätigen lassen.

Wenn Sie die Programme, die die einzelnen Verarbeitungsschritte realisieren, aneinanderketten, so sorgen Sie dafür, dass der Gesamt-Vorgang in der vorgesehenen Reihenfolge bearbeitet wird. Teilprogramme ketten Sie über Angaben im PEND-Aufruf. Im PEND-Aufruf wählen Sie im Feld KCOM die Operationsmodifikation RE, wenn Sie zum Ende des Verarbeitungsschrittes auch einen Sicherungspunkt setzten wollen, und nennen im Feld KCRN den Transaktionscode des Folgeprogramms. Wenn Sie nur den Verarbeitungsschritt beenden wollen, jedoch keinen Sicherungspunkt setzen, wählen Sie beim PEND-Aufruf statt RE die Modifikation KP. Empfängt openUTM die nächste Eingabe-Nachricht, so wird das Folgeprogramm gestartet.

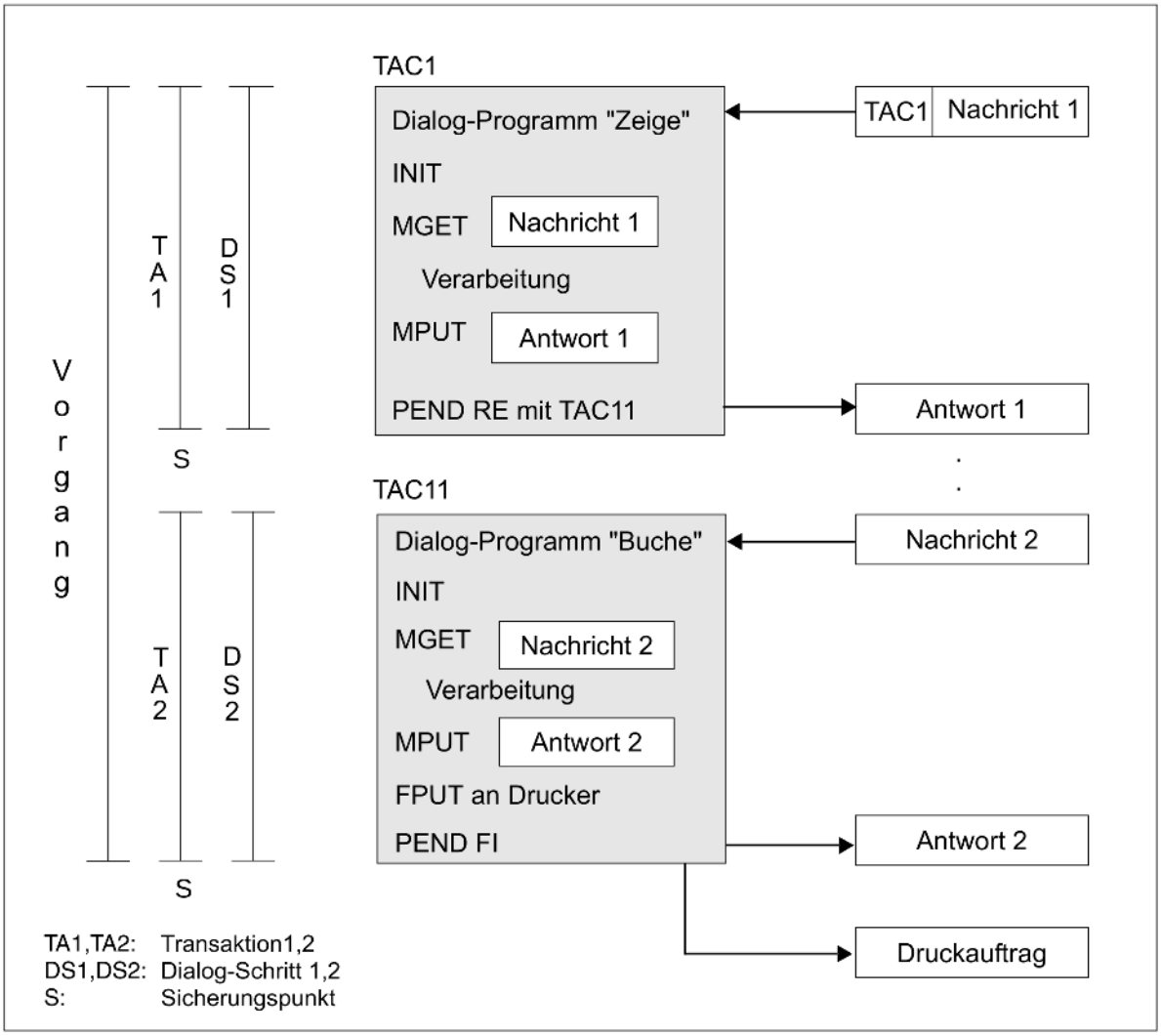


Bild: Mehrschritt-Vorgang

Der im Bild dargestellte Vorgang umfasst zwei Teilprogramme, die jeweils einen Dialog-Schritt realisieren. Sie können jedoch beliebig viele Teilprogramme zu einem Vorgang zusammenfassen. openUTM stellt allen Teilprogrammen eines Vorgangs spezifische Speicherbereiche zur Verfügung, in denen die Teilprogramme Informationen weitergeben können (siehe "Die KDCS-Speicherbereiche bei openUTM"). Diese Speicherbereiche sind in die Transaktionssicherung einbezogen.

Das zweite Teilprogramm enthält neben dem MPUT-Aufruf auch einen FPUT-Aufruf. Dieser Aufruf erzeugt keine Dialog- sondern eine Asynchron-Nachricht, in diesem Fall eine Ausgabenachricht an einen Drucker.

Gleicher Verarbeitungsschritt für unterschiedliche Aktionen

Laufen in einer Anwendung mehrere gleichartige Aktionen ab, so liegt es nahe, dass Verarbeitungsschritte, die bei allen Aktionen gleich sind, in einem einzigen Teilprogramm verarbeitet werden. Dies wird anhand der Mehrschritt-Vorgänge des folgenden Beispiels veranschaulicht.

Zu Beginn verschiedener Aktionen sollen die Daten eines Versicherungsvertrags gezeigt werden, um anschließend einen der folgenden Schritte auszuführen:

- Daten ändern
- Daten löschen
- Prämie berechnen
- einen Schaden melden

Der erste Schritt ist für alle 4 Aktionen gleich, die weiteren Schritte sind unterschiedlich (siehe nachfolgendes Bild).

Alle 4 Aktionen werden zunächst in dem gleichen Teilprogramm bearbeitet. Zu jeder Aktion gehört ein eigener Transaktionscode: Dem ersten Teilprogramm sind also vier unterschiedliche Transaktionscodes zugeordnet. Nach Start des Teilprogramms zeigt openUTM den Transaktionscode, der für den Vorgangsstart verwendet wurde, im Feld KCTACVG/ kccv_tac des KB-Kopfes an. Das Teilprogramm legt dann in Abhängigkeit vom verwendeten Transaktionscode fest, welches Teilprogramm als Folgeteilprogramm gestartet werden soll, d.h. welcher Folge-TAC beim PEND-Aufruf ins Feld KCRN eingetragen werden soll.

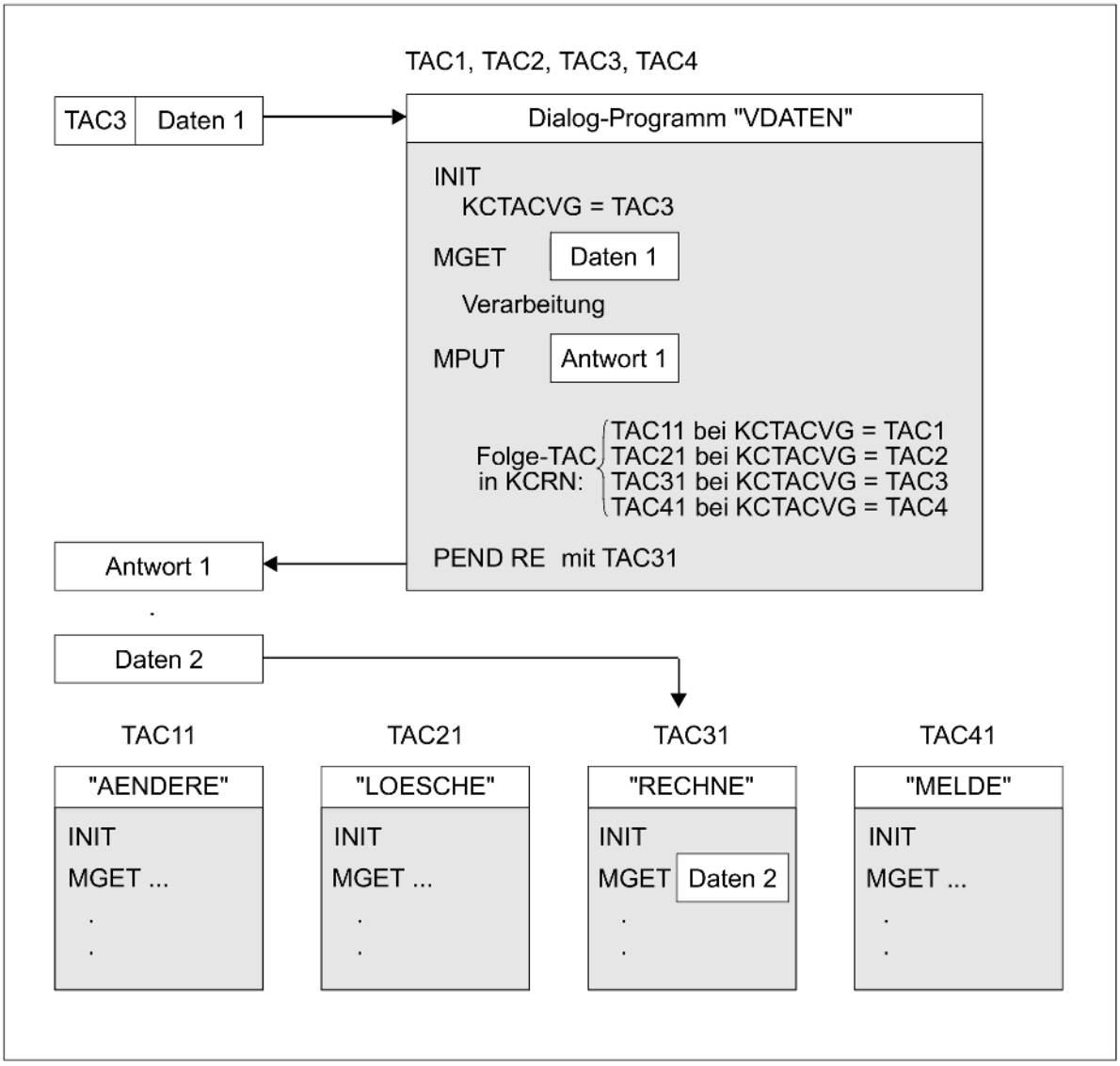


Bild: Teilprogramm, dem mehrere Transaktionscodes zugeordnet sind

2.3.2 Mehrere Teilprogramme in einem Verarbeitungsschritt

Wenn verschiedene Verarbeitungsschritte gleiche Aufgabenteile enthalten, dann ist es sinnvoll, die einzelnen Verarbeitungsschritte in mehrere Teile zu zerlegen. Für gleiche Aufgabenteile schreibt man jeweils ein Teilprogramm, das von allen Verarbeitungsschritten gemeinsam verwendet wird. Das Grundschema des Vorgangs-Aufbaus "ein Verarbeitungsschritt = ein Teilprogramm" wird also variiert: **Ein** Verarbeitungsschritt wird durch **mehrere** Teilprogramme realisiert. Dies ermöglichen die Aufrufe PEND PR/PA/SP, die das Teilprogramm beenden, nicht jedoch den Verarbeitungsschritt. Bei PEND SP wird ein Sicherungspunkt gesetzt, bei PEND PA/PR bleibt die Transaktion offen.

Die beiden Vorgänge im folgenden Bild sind - obwohl Sie jeweils drei Teilprogrammläufe umfassen - Einschritt-Vorgänge:

- Der Vorgang, dem der Transaktionscode TAC1 zugeordnet ist, hat die Aufgabe, die Daten eines Versicherungsvertrags zu ändern.
- Der Vorgang, dem der Transaktionscode TAC2 zugeordnet ist, hat die Aufgabe, die Daten eines Versicherungsvertrags zu löschen.

Gemeinsam ist beiden Aufgaben, dass der ursprünglich gespeicherte Datensatz verändert werden muss. Um diesen gemeinsamen Teil auch in einem eigenem Teilprogramm realisieren zu können, wird der Verarbeitungsschritt in einzelne Teile zerlegt.

Das Feld KCTACVG enthält nach dem INIT-Aufruf den TAC, mit dem der Vorgang gestartet wurde. Das Teilprogramm "ZENTRAL" entscheidet daraufhin, mit welchem Teilprogramm die Transaktion fortgesetzt wird. Den TAC dieses Teilprogramms geben Sie im Feld KCRN an, und zwar sowohl beim MPUT als auch beim PEND PR. Die MPUT-Nachricht wird nicht an den Kommunikationspartner, sondern an das Folge-Teilprogramm gesendet und wird dort mit MGET gelesen. Da im Teilprogramm "ZENTRAL" der Verarbeitungsschritt nicht abgeschlossen wird, ist ein MPUT-Aufruf in diesem Teilprogramm nicht zwingend erforderlich: Die Daten könnten statt mit MPUT auch über Vorgangs-spezifische Speicher übergeben werden (siehe auch Abschnitt „Die KDCS-Speicherbereiche bei openUTM“).

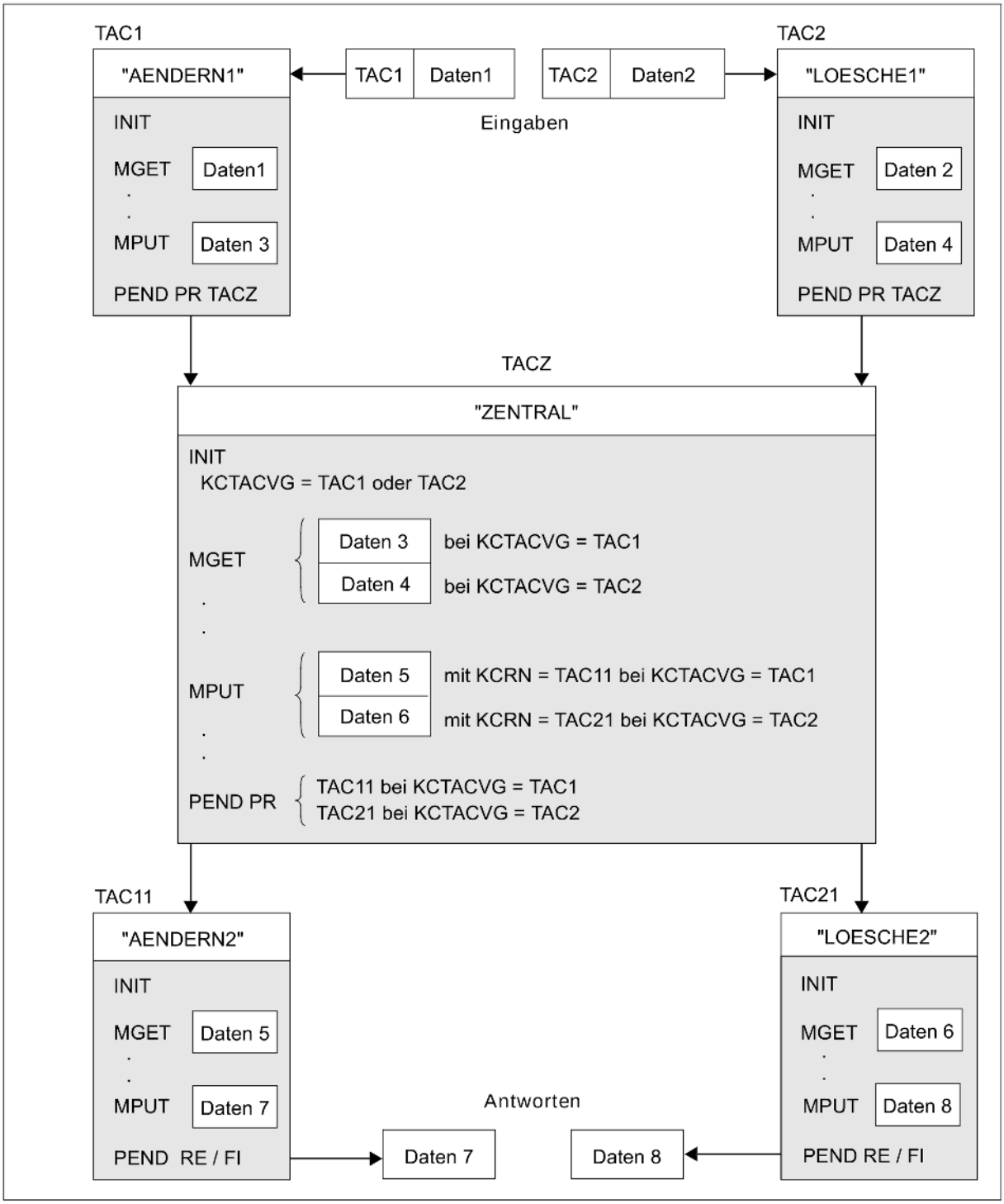


Bild: Mehrere Teilprogramme realisieren einen Verarbeitungsschritt

2.3.3 Mehrere Verarbeitungsschritte in einem Teilprogramm

Ein UTM-Teilprogramm läuft zwischen INIT und PEND in einem Prozess ab. Da der Prozess am Teilprogramm-Ende freigegeben wird, läuft ein Folge-Teilprogramm möglicherweise in einem anderen Prozess ab als das vorhergehende. Das führt dazu, dass das Folge-Teilprogramm nicht auf die Prozess-spezifische Umgebung (Kontext) - wie Betriebsmittel und Programm-spezifische Datenbereiche - des ersten Teilprogramms zugreifen kann. Dies ist auch in der Regel nicht notwendig, da Daten über einen an das Folge-Teilprogramm gerichteten MPUT-Aufruf oder über von openUTM zur Verfügung gestellte Vorgangs-spezifische Speicherbereiche weitergereicht werden können.

openUTM bietet jedoch auch eine Möglichkeit, den Prozess-spezifischen Kontext über mehrere Verarbeitungsschritte hinweg zur erhalten: Hierzu verwenden Sie den Aufruf PGWT (Program Wait) mit den Modifikationen KP, PR, CM und RB. Damit wird ein Wartepunkt gesetzt, ohne das Teilprogramm zu beenden, d.h. der Programmkontext bleibt erhalten. Diese Varianten werden zu unterschiedlichen Zwecken eingesetzt:

- PGWT KP beendet den Verarbeitungsschritt und sendet die MPUT-Nachricht. Das Teilprogramm wartet so lange, bis die nächste Nachricht vom Partner eintrifft. PGWT KP entspricht dem PEND KP-Aufruf im vorhergehenden und dem INIT-Aufruf im nachfolgenden Teilprogramm.
- PGWT PR wartet auf eine Nachricht an eine Queue, ohne den Verarbeitungsschritt zu beenden. PGWT PR entspricht einem PEND PA/PR-Aufruf im vorhergehenden und einem INIT-Aufruf im nachfolgenden Teilprogramm.
- PGWT CM beendet die Transaktion, ohne das Teilprogramm zu beenden. Ein mit PGWT CM gesetzter Sicherungspunkt ist jedoch kein Wiederanlaufpunkt. Die Folge-Transaktion kann deshalb nicht mit PEND RS zurückgesetzt werden, sondern nur mit PGWT RB.

  Außerdem gilt:

  - Wurde vor dem PGWT CM ein MPUT-Aufruf ausgeführt, so wird die MPUT-Nachricht gesendet und das Teilproprogramm wartet solange, bis eine Antwort vom Partner eintrifft. Dieser PGWT CM entspricht einem PEND RE-Aufruf im vorhergehenden und einem INIT-Aufruf im nachfolgenden Teilprogramm.
  - Wurde vor dem PGWT CM dagegen kein MPUT-Aufruf ausgeführt, wird das Teilprogramm sofort fortgesetzt. Der PGWT CM ohne vorhergehenden MPUT-Aufruf entspricht einem PEND SP-Aufruf im vorhergehenden und einem INIT-Aufruf im nachfolgenden Teilprogramm.
- PGWT RB setzt eine Transaktion zurück.

Die Funktionen zweier aufeinanderfolgender Teilprogramme können somit von einem Teilprogramm übernommen werden. Damit läuft die gesamte Funktionsfolge unter ein und demselben Prozess ab. Der Prozess-spezifische Kontext steht bis zum Ende des Teilprogrammlaufs zur Verfügung.

Das Teilprogramm belegt in dieser Zeit einen Prozess exklusiv. Dadurch benötigt die UTM-Anwendung in der Regel mehr Betriebsmittel (Prozesse).

Programmsysteme, die eine kombinierte SEND/RECEIVE-Schnittstelle erwarten, können mit Hilfe des PGWT-Aufrufs leicht unter openUTM zum Ablauf gebracht werden.

Neben den Aufrufen PEND PR/PA/SP bietet der PGWT-Aufruf somit eine weitere Möglichkeit, das verarbeitungsschritt-modulare Aufbau-Schema zu variieren:
Während es die Aufrufe **PEND PR/PA/SP** ermöglichen, **einen** Verarbeitungsschritt auf **mehrere** Teilprogramme aufzuteilen, erlauben es **PGWT**-Aufrufe, **mehrere** Verarbeitungsschritte in **einem** Teilprogramm zu realisieren.

Da mit PGWT-Aufrufen wertvolle Betriebsmittel gebunden werden, sollten diese Aufrufe in einer Anwendung sparsam eingesetzt werden; er sollte **nur** dort verwendet werden, wo die anderen Möglichkeiten der KDCS-Schnittstelle nicht ausreichen; eine häufige Verwendung von PGWT-Aufrufen kann die Performance einer UTM-Anwendung negativ beeinflussen.

2.3.4 Unterprogramm-Aufrufe aus Teilprogrammen

Sie können in einem Teilprogramm auch Unterprogramm-Aufrufe absetzen, z.B. C/C++-Funktionen oder COBOL-Unterprogramme. Diese Unterprogramme dürfen selbst wieder Unterprogrammaufrufe enthalten. Bei Unterprogrammaufrufen werden die Programme über ihren Programmnamen (in C/C++: Funktionsnamen) und nicht über einen Transaktionscode aufgerufen.

Unterprogramme können auch in einer anderen Programmiersprache erstellt werden wie das rufende Programm.

Was auf BS2000-Systemen zu beachten ist und welche Compiler und Laufzeitsysteme dazu benötigt werden, ist im openUTM-Handbuch „Einsatz von UTM-Anwendungen auf BS2000-Systemen“ beschrieben.

Mehr zu Unterprogrammaufrufen finden Sie in Abschnitt „C/C++-Teilprogramme als Unterprogramme“ und Abschnitt „COBOL-Teilprogramm als Unterprogramm“.

Der Programmlauf muss dabei entweder wieder in das Teilprogramm zurückführen oder in einem Unterprogramm durch einen PEND-Aufruf beendet werden.

2.3.5 Vorgänge ketten

Im Standardfall werden Vorgänge mit dem Aufruf **PEND FI** (finish) beendet. Beim PEND FI wird eine Dialog-Nachricht an das Terminal, Client-Programm oder eine andere Anwendung ausgegeben. Das Vorgangsende ist also zugleich Verarbeitungsschritt-Ende.

Sie können aber mit dem Aufruf **PEND FC** (finish and continue) einen weiteren Vorgang anketten und den Verarbeitungsschritt dort fortsetzen. Dies ist z.B. dann sinnvoll, wenn zum Vorgangsende keine Dialog-Nachricht an den Client ausgegeben werden soll oder wenn dem Benutzer die Verwendung von TACs verborgen bleiben soll.

Beim PEND FC werden wie beim PEND FI Transaktion und Vorgang beendet, sowie Vorgangs-spezifische Speicherbereiche freigegeben (LSSBs, KB). Daten an den angeketteten Vorgang werden mit einem MPUT-Aufruf weitergegeben, Vorgangs-spezifische Speicherbereiche können Sie hierfür nicht verwenden.

Beim Programmieren gibt es die folgenden Unterschiede zum PEND FI:

- Im Feld KCRN müssen Sie beim PEND FC den TAC des angeketteten Vorgangs angeben.
- Vor PEND FC ist kein MPUT notwendig. Falls aber ein MPUT aufgerufen wird, dann muss in KCRN ebenfalls der Folge-TAC eingetragen werden (wie bei PEND PA/PR).

Wird die erste Transaktion des angeketteten Vorgangs zurückgesetzt, dann setzt openUTM auf dem Sicherungspunkt bei PEND FC auf und startet den angeketteten Vorgang erneut.

2.3.6 Vorgänge kellern

Ein Benutzer am Terminal kann einen Vorgang kellern, d.h., er kann einen begonnenen Vorgang unterbrechen, einen anderen Vorgang einschieben und nach dessen Beendigung den unterbrochenen Vorgang fortsetzen. Ein Vorgang kann nur gekellert werden, wenn er sich auf einem Sicherungspunkt befindet, d.h. direkt nach einem PEND RE.
Dafür gibt es zwei Möglichkeiten:

- durch Drücken einer mit SFUNC ...,STACK=... generierten Funktionstaste
- durch Nutzung des Event-Exits INPUT

Gekellerten Vorgang fortsetzen

Ein gekellerter Vorgang, auch Vorgänger genannt, wird aktiviert, sobald sich der eingeschobene Vorgang mit PEND FI beendet hat. Der eingeschobene Vorgang erzeugt dabei eine Ausgabenachricht. Da openUTM eine Vorgangs-Kellerung nur auf einem Sicherungspunkt erlaubt, stehen sowohl die letzte Ausgabenachricht als auch die Vorgangs-spezifischen Bereiche (KB, LSSB) des Vorgängers noch zur Verfügung. Wie nun die Ausgabe-Nachrichten der beiden Vorgänge behandelt werden, bestimmen Sie beim (letzten) MPUT-Aufruf im eingeschobenen Vorgang.
Dabei stehen drei Möglichkeiten zur Verfügung:

- MPUT NE im eingeschobenen Vorgang gibt die Nachricht dieses Vorgangs zusammen mit der Meldung K096 auf dem Bildschirm aus. Nach Drücken der Eingabe-Taste erhält der Benutzer die letzte Ausgabenachricht des Vorgängers.
- MPUT PM mit KCLM = 0 gibt sofort die letzte Ausgabenachricht des Vorgängers aus (PM steht für "predecessor message").
- MPUT PM mit KCLM > 0 (nur im Format-Betrieb erlaubt) überschreibt die Ausgabenachricht des Vorgängers mit der Nachricht des eingeschobenen Vorgangs, und zwar bis zu der in KCLM angegebenen Länge. Anschließend gibt openUTM die so überarbeitete Nachricht des Vorgängers aus. Das bei MPUT PM angegebene Format sollte ein Teilformat des Vorgänger-Formats sein.

MPUT PM ist im eingeschobenen Vorgang nur erlaubt, wenn das Teilprogramm mit PEND FI beendet wird, also nur im letzten Verarbeitungsschritt des Vorgangs.

Vorgangs-Stapel

Durch Kellerung von Vorgängen entstehen Vorgangs-Stapel.
Sie haben folgende Möglichkeiten, sich Informationen über den aktuellen Vorgangs-Stapel zu beschaffen.

- Die Felder KCHSTA und KCDSTA des KB-Kopfes zeigen die Höhe des Stapels und dessen Veränderung seit dem letzten Teilprogrammlauf an.
- Der INFO-Aufruf INFO PC (predecessor conversation) informiert über den direkten Vorgänger im Stapel.

Durch PEND FI in einem eingeschobenen Vorgang kehrt man immer zum unmittelbaren Vorgänger zurück; ein Stapel wird dann aufgelöst, wenn sich der letzte eingeschobene Vorgang beendet hat.

2.4 Message Queuing (Asynchron-Verarbeitung)

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Queues ausgetauscht werden. Da zwischen dem Senden und Empfangen der Nachrichten keine zeitliche Synchronisation besteht, werden Nachrichten an Message Queues auch **Asynchron-Nachrichten** genannt**.** Die Kommunikation wird in Form von **Asynchron-Aufträgen** abgewickelt. Ein Asynchron-Auftrag besteht aus der Asynchron-Nachricht, dem Empfänger der Nachricht und ggf. dem gewünschten Ausführungszeitpunkt.

openUTM bietet Ihnen zwei Arten von Message Queues:

- **UTM-gesteuerte Queues**:
  Bei UTM-gesteuerten Queues wird der zwischengeschaltete Queuing-Mechanismus komplett von openUTM zur Verfügung gestellt, d.h. openUTM übernimmt neben der reinen Queuing-Funktionalität auch die weitere Verarbeitung der Nachricht wie z.B. die Ausgabe an einen Kommunikationspartner oder das Starten eines Vorgangs.
- **Service-gesteuerte Queues**:
  Bei Service-gesteuerten Queues ist ein Vorgang (= Service) verantwortlich für die Weiterverarbeitung der Nachricht, d.h. openUTM führt nur eine reine Queuing-Funktionalität aus. Der Kommunikationspartner, für den die Nachricht letztendlich bestimmt ist, muss selbständig eine Nachricht aus einer Queue lesen (KDCS-Aufruf DGET). Falls in der Queue keine Nachricht vorliegt, dann kann ein Vorgang auch auf das Eintreffen einer Nachricht warten.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen wird Message Queuing nur Knoten-lokal unterstützt, d.h. Asynchron-Nachrichten können nur in der Knoten-Anwendung ablaufen, gelesen, angezeigt oder administriert werden, in der diese auch erzeugt wurden.
Einzige Ausnahme: Mit dem Online-Import können Sie Asynchron-Aufträge einer beendeten Knoten-Anwendung in eine laufende Knoten-Anwendung übernehmen.

> ¡ Allgemeine Informationen über das Message Queuing Konzept und dessen Einsatzmöglichkeiten finden Sie im openUTM-Handbuch „Konzepte und Funktionen“.

2.4.1 Dead Letter Queue

Die **Dead Letter Queue** ist eine TAC-Queue mit dem festem Namen KDCDLETQ. Sie steht immer zur Verfügung, um Asynchron-Nachrichten an Transaktionscodes, TAC-Queues, LPAPs oder OSI-LPAPs zu sichern, die nicht verarbeitet werden konnten.

Die Sicherung von Asynchron-Nachrichten in der Dead Letter Queue kann in der Generierung durch den Parameter DEAD-LETTER-Q der TAC-, LPAP- und OSI-LPAP-Anweisung für jedes Nachrichtenziel einzeln ein- und ausgeschaltet werden.

Um die Nachrichten in der Dead Letter Queue evtl. nach einer Fehlerbehebung noch verarbeiten zu können, müssen sie entweder ihrem ursprünglichen Ziel oder einem neuen Ziel zugeordnet werden. Mit DADM MV kann eine einzelne Nachricht, mit DADM MA können alle Nachrichten der Dead Letter Queue in eine bestimmte oder die ursprünglichen Queues verschoben werden. Dabei ist zu beachten, dass die Nachrichten immer nur an ein Ziel vom Typ des ursprünglichen Ziels verschoben werden können (TAC >> TAC, LPAP >> LPAP, OSI-LPAP >> OSI-LPAP).

Das Nachrichtenaufkommen der Dead Letter Queue kann mit der K134-Meldung überwacht werden (siehe openUTM-Handbuch „Anwendungen generieren“, Operand DEAD-LETTER-Q-ALARM der MAX-Anweisung).

Pagepool-Engpass vermeiden

Da die Dead Letter Queue im Pagepool abgelegt wird, sollte der Pagepool für die Sicherung von Nachrichten in der Dead Letter Queue ausreichend groß generiert werden.

Um einen Pagepool-Engpass zu vermeiden oder zu beheben, sind folgende Maßnahmen möglich:

- Nachrichten aus der Dead Letter Queue verschieben (DADM MV/MA).
- Nachrichten aus der Dead Letter Queue löschen (DADM DL/DA).
- Durch Generierung von QLEV die maximale Anzahl der Nachrichten der Dead Letter Queue beschränken.
- Dead Letter Queue zeitweise sperren (STATUS = OFF).
- Die Sicherung von Nachrichten in der Dead Letter Queue für einzelne Ziele zeitweise ausschalten, z.B. über WinAdmin oder WebAdmin oder über programmierte Administration.

2.4.2 Nachrichten an UTM-gesteuerte Queues

Bei UTM-gesteuerten Queues kontrolliert openUTM die weitere Verarbeitung von Nachrichten, die ein Teilprogramm in UTM-gesteuerte Queues schreibt. D.h. beim Senden einer Nachricht an eine UTM-gesteuerte Queue steht fest, wie die Nachricht weiterverarbeitet werden soll. Daher klassifiziert man die mit dieser Nachricht verknüpften Aufträge nach dem Empfänger:

- Ausgabe-Aufträge
- Hintergrund-Aufträge

  Bei den Hintergrund-Aufträgen kann weiter differenziert werden:

  - **lokale** Hintergrund-Aufträge
    d.h. Hintergrund-Aufträge, die Services der eigenen Anwendung anfordern
  - Hintergrund-Aufträge, die **ferne** Services anfordern
    Auf diese Spezifika wird im Abschnitt „UTM-gesteuerte Queues bei verteilter Verarbeitung“ näher eingegangen.

2.4.2.1 Ausgabe-Aufträge

Ausgabe-Aufträge sind Asynchron-Aufträge, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker oder an ein Terminal auszugeben. Ausgabeziel kann aber auch eine andere Anwendung sein, die über die Transportsystem-Schnittstelle angeschlossen wurde.

Ausgabe-Aufträge setzen sich zusammen aus der Angabe des Ausgabeziels und der Asynchron-Nachricht, die ausgegeben werden soll.

Ausgabe-Aufträge werden durch entsprechende MQ-Aufrufe aus einem Teilprogramm der UTM-Anwendung ausgelöst.

2.4.2.2 Hintergrund-Aufträge

Hintergrund-Aufträge sind Asynchron-Aufträge, die an einen Asynchron-Vorgang der eigenen oder einer fernen Anwendung gerichtet sind. Hintergrund-Aufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluss auf den aktuellen Dialog hat.

Hintergrund-Aufträge setzen sich zusammen aus dem Transaktionscode (TAC) des Teilprogramms, mit dem der Hintergrund-Auftrag gestartet wird (Vorgangs-TAC), und ggf. einer Nachricht für das Teilprogramm. Dabei bestimmt der Typ des Transaktionscodes, dass der Auftrag asynchron - und nicht als Dialog-Auftrag - verarbeitet wird.

Hintergrund-Aufträge können wie folgt erzeugt werden:

- Durch Eingabe von einem Terminal
- Durch einen MQ-Aufruf aus einem Vorgang der eigenen UTM-Anwendung
- Durch eine Nachricht von einer anderen Anwendung, die mit der UTM-Anwendung über das LU6.1-, LU6.2- oder OSI TP-Protokoll kommuniziert
- Durch Eingabe von einer anderen Anwendung, die über die Transportsystemschnittstelle angeschlossen ist
- Durch eine UTM-Meldung, wenn dieser Meldung das Meldungsziel MSGTAC (also ereignisgesteuert, siehe "Asynchron-Vorgang MSGTAC") oder als Benutzer-spezifisches Meldungsziel der TAC eines Asynchron-Vorgangs zugeordnet ist

Hintergrundaufträge können nach abnormalem Beenden eines Vorgangs erneut gestartet werden (Redelivery), siehe "Redelivery bei Hintergrundaufträgen" oder in die Dead Letter Queue gestellt werden.

2.4.2.3 MQ-Aufrufe der KDCS-Schnittstelle

openUTM stellt funktionell mächtige, aber einfach zu programmierende Aufrufe für UTM-gesteuerte Queues zur Verfügung. Der Zusatz "free" in den Aufrufnamen soll widerspiegeln, dass es sich beim Message Queuing um eine vom Sender entkoppelte und von der Verfügbarkeit des Empfängers unabhängige Art der Kommunikation handelt.

- FPUT (**F**ree message **PUT**)
  FPUT-Aufrufe dienen zum Senden von Asynchron-Nachrichten. Das Ziel kann ein Ausgabegerät sein (Ausgabe-Auftrag) oder ein Asynchron-Vorgang (Hintergrund-Auftrag), oder eine Anwendung, siehe "UTM-gesteuerte Queues bei verteilter Verarbeitung".
  Eine Asynchron-Nachricht kann auch aus mehreren Teilnachrichten bestehen. Für jede Teilnachricht ist dann ein eigener FPUT-Aufruf notwendig.
- DPUT (**D**elayed free message **PUT**)
  Auch mit dem DPUT-Aufruf wird eine Asynchron-Nachricht oder -Teilnachricht an ein Ausgabegerät, einen Asynchron-Vorgang oder eine andere Anwendung gesendet, siehe "UTM-gesteuerte Queues bei verteilter Verarbeitung". Der DPUT-Aufruf bietet aber gegenüber dem FPUT-Aufruf zusätzlich die Möglichkeit der Zeitsteuerung oder der Verwendung von Quittungsaufträgen.
- FGET (**F**ree message **GET**)
  Der Aufruf FGET dient zum Lesen von Asynchron-Nachrichten oder -Teilnachrichten innerhalb eines Asynchron-Vorgangs.
- MCOM (**M**essage **COM**plex)
  Der Aufruf MCOM dient dazu, einem Asynchron-Auftrag Quittungsaufträge zuzuordnen.
- DADM (**D**elayed free message **ADM**inistration)
  Mit DADM können Übersichtsinformationen über den gesamten Inhalt einer Queue oder gezielt über einzelne Elemente angefordert werden. Außerdem lässt sich mit DADM die Bearbeitungsreihenfolge steuern: Sie können Aufträge vorziehen, einzelne Aufträge stornieren oder auch die gesamte Queue löschen.

Das genaue Format dieser Aufrufe und weitere Informationen finden Sie im Kapitel „KDCS-Aufrufe“.

2.4.2.4 Aufbau eines Asynchron-Vorgangs

Ein Asynchron-Vorgang beginnt mit einem Asynchron-Teilprogramm. Diesem Teilprogramm wurde bei der Generierung ein Transaktionscode mit TYPE=A (asynchronous) zugeordnet. Ein Asynchron-Teilprogramm unterscheidet sich von einem Dialog-Teilprogramm nicht nur durch den Typ des Transaktionscodes, sondern ist auch anders aufgebaut.

Aufbau eines Asynchron-Teilprogramms

Asynchron-Programme müssen weder eine Eingabenachricht lesen noch eine Ausgabenachricht erzeugen. Nach dem INIT kann im ersten Teilprogramm des ersten Verarbeitungsschritts ein FGET-Aufruf folgen, mit dem die Asynchron-Nachricht des Erzeugers des Hintergrund-Auftrags gelesen werden kann. Das kann sein:

- eine vollständige Nachricht
- oder eine Teilnachricht
- oder eine leere Nachricht, falls der Auftrag nur aus einem TAC besteht (z.B. über eine Funktionstaste des Terminals erzeugt).

Danach können - mit Ausnahme von MGET - beliebige KDCS-Aufrufe folgen. Ein MGET-Aufruf ist jedoch in einem Folge-Teilprogramm oder Folge-Verarbeitungsschritt erlaubt.

Es ist nicht möglich, mit MPUT eine Antwort an den Kommunikationspartner, von dem die Asynchron-Nachricht gesendet wurde, zu senden. Da der Vorgang entkoppelt vom Partner abläuft, ist der Partner zum Zeitpunkt der Bearbeitung u.U. gar nicht mehr mit der Anwendung verbunden. Dem Partner kann jedoch mit FPUT oder DPUT eine Nachricht gesendet werden, die dann in die dem Partner zugeordnete Message Queue eingetragen wird. Ein MPUT-Aufruf ist nur erlaubt, wenn die Nachricht an ein Folge-Teilprogramm oder an einen Auftragnehmer-Vorgang gerichtet ist.

Letzter UTM-Aufruf in Ihrem Teilprogramm muss ein PEND sein, wie im Abschnitt „Programmrahmen“ beschrieben. Ein Asynchron-Programm wird im Standardfall mit einem PEND FI-Aufruf beendet. Damit ist auch der Vorgang beendet. openUTM übermittelt evtl. zu sendende Nachrichten dem oder den Partnern.
Weitere PEND-Varianten sind möglich, z.B. PEND PA/PR/SP zur Teilprogrammkettung, und PEND KP und RE für verteilte Verarbeitung. Auch die Aufrufe PGWT KP, CM und PR können Sie in einem Asynchron-Teilprogramm nutzen, z.B. PGWT KP, wenn Sie bei verteilter Verarbeitung nur den Verarbeitungsschritt, nicht aber Teilprogramm und Transaktion beenden wollen.

Ein Asynchron-Vorgang kann in mehrere Teilprogrammläufe, und - bei verteilter Verarbeitung - auch in mehrere Verarbeitungsschritte gegliedert werden (siehe "Asynchron-Vorgang aus mehreren Teilprogrammen" bzw. "Asynchron-Vorgänge, die ihrerseits Aufträge absetzen").

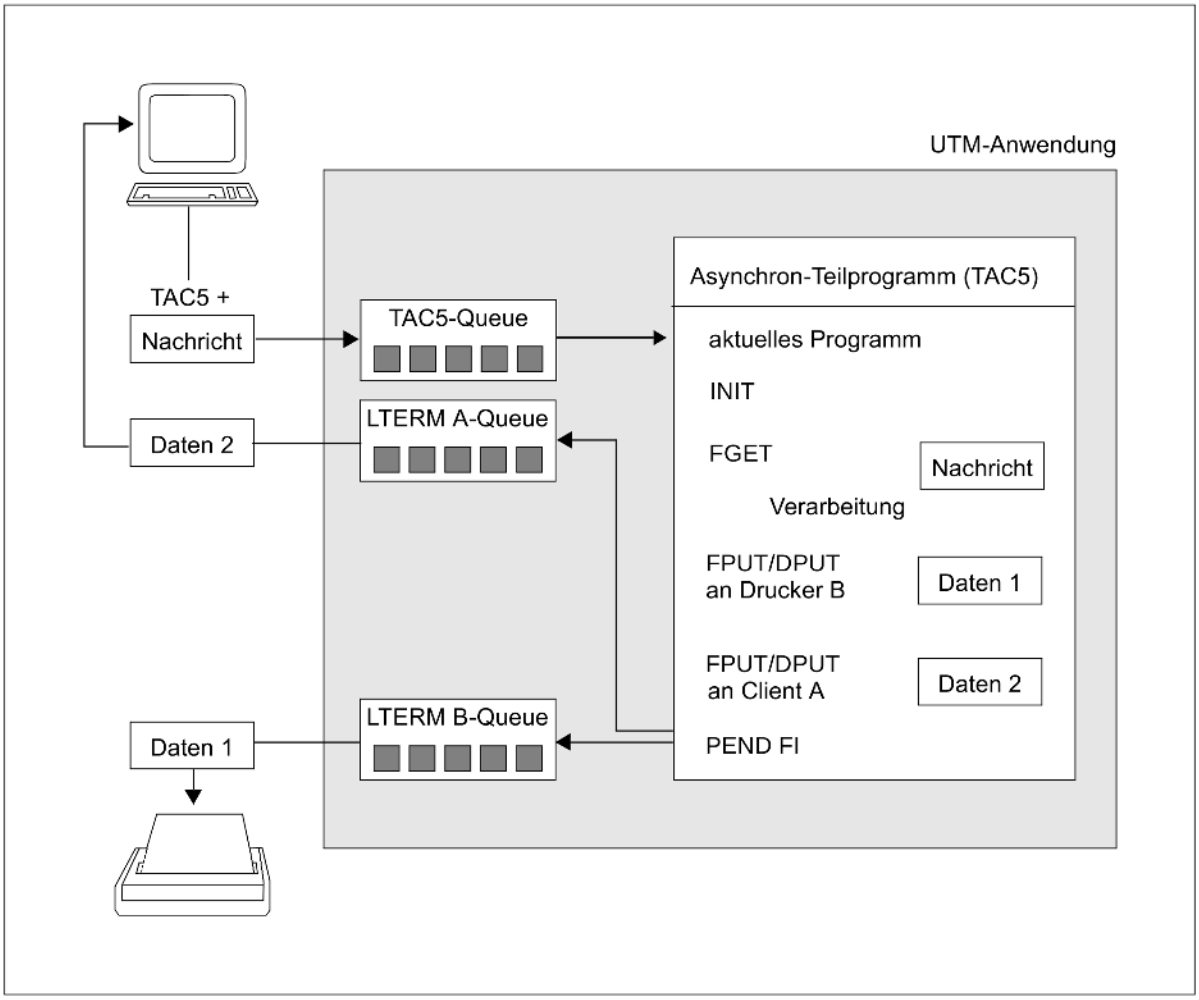


Bild: Aufbau eines Asynchron-Teilprogramms

Kombiniertes Dialog- und Asynchron-Programm

Einem Teilprogramm können mehrere Transaktionscodes zugeordnet werden. Dabei ist es auch möglich, dem Teilprogramm sowohl einen Dialog-TAC als auch einen Asynchron-TAC zuzuordnen. Das Programm kann also im Dialog oder asynchron gestartet werden. openUTM zeigt dies nach Vorgangs-Start im Feld KCPRIND des KB-Kopfes an: "A" für asynchron, "D" für Dialog. Das Teilprogramm kann dieses Feld auswerten und entsprechend verzweigen.

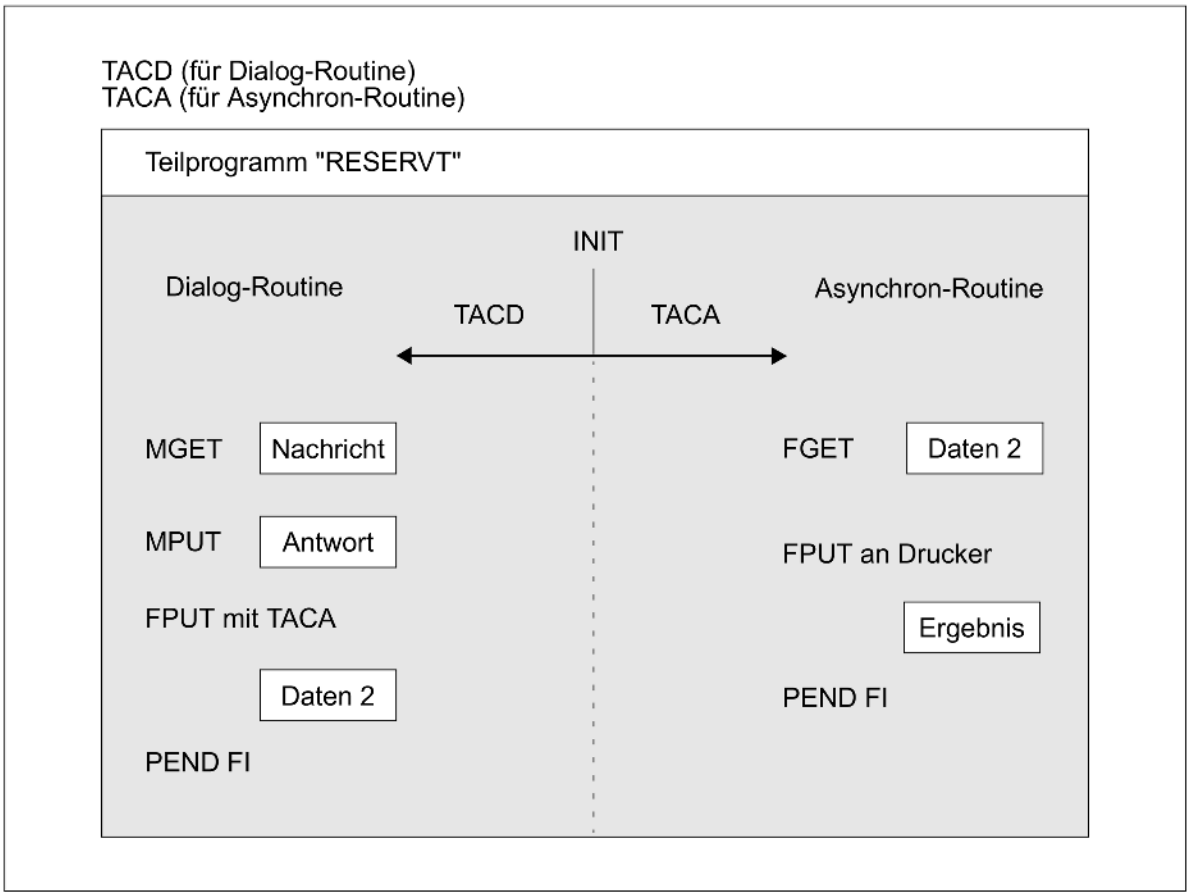


Bild: Dialog- und Asynchron-Verarbeitung in einem Programm

Wird das im Bild dargestellte Programm mit TACD gestartet, wird nach dem PEND der FPUT ausgeführt, wodurch das gleiche Teilprogramm noch einmal gestartet wird, um einen unabhängigen 2. Vorgang zu bearbeiten. Für ein Dialog-Programm bedeutet die Auslagerung eines Asynchron-Vorgangs u.U. bessere Antwortzeiten für den Dialog mit dem Client.

Asynchron-Vorgang aus mehreren Teilprogrammen

Ein Asynchron-Vorgang kann in mehrere Teilprogramme und Transaktionen gegliedert sein. Das letzte Teilprogramm wird mit PEND FI abgeschlossen. In den vorhergehenden Teilprogrammen sind die PEND-Varianten SP oder PA/PR möglich.

PEND-Varianten, die einen Verarbeitungsschritt beenden, wie PEND KP oder PEND RE, sind innerhalb eines Asynchron-Vorgangs nur bei verteilter Verarbeitung möglich (vgl. das Bild auf der folgenden Seite).

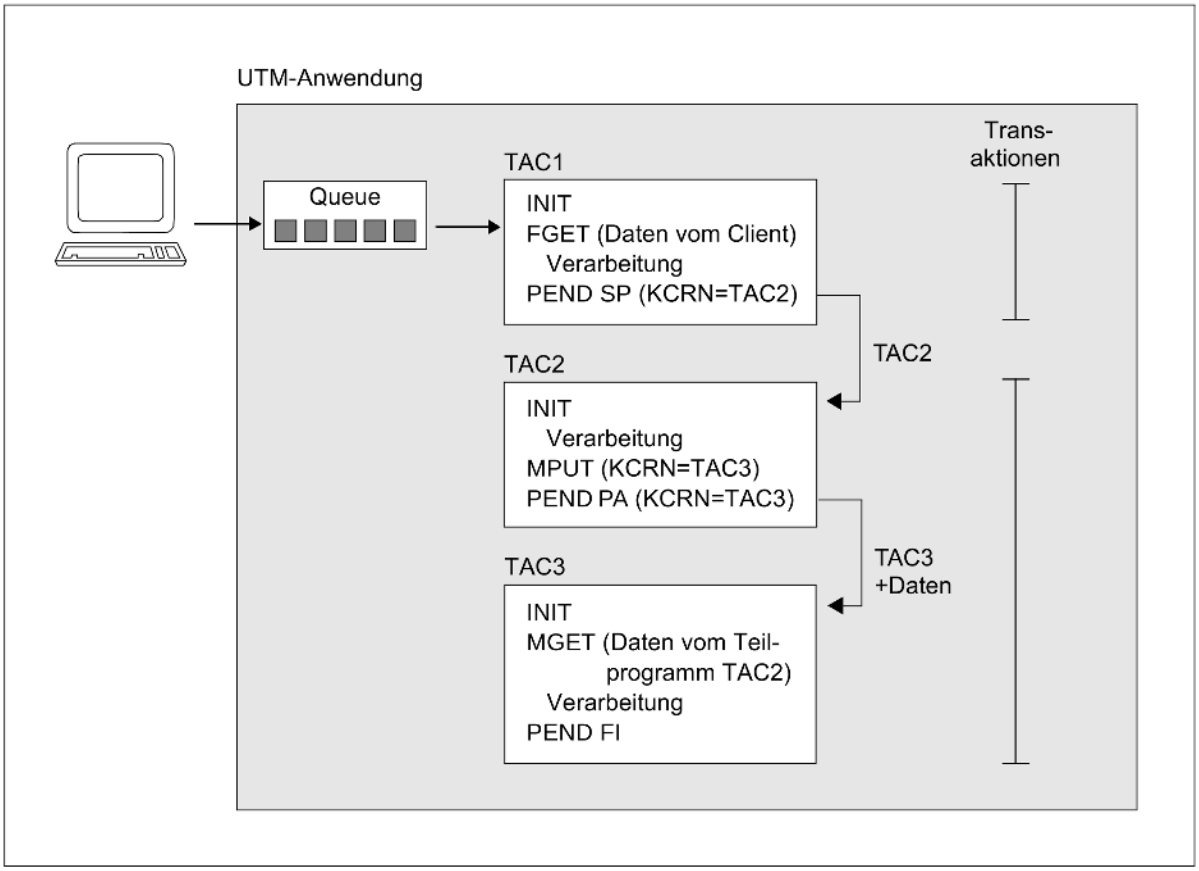


Bild: Struktur eines in drei Teilprogramme gegliederten Asynchron-Vorgangs

Im dargestellten Beispiel wird von einem Terminal aus ein Hintergrund-Auftrag an einen Asynchron-Vorgang der eigenen Anwendung gestellt. Hierzu wird am Terminal der Transaktionscode dieses Vorgangs und ggf. eine Nachricht eingegeben. openUTM reiht den Auftrag automatisch in die entsprechende Queue ein und startet den Asynchron-Vorgang entkoppelt vom Auftraggeber, sobald die notwendigen Betriebsmittel zur Verfügung stehen. Das erste Teilprogramm liest die Daten mit FGET ein und schließt mit PEND SP. Es wird ein Sicherungspunkt gesetzt und das in KCRN angegebene Folge-Teilprogramm gestartet.

Das Programm TAC1 hat keinen MPUT an das Programm TAC2 übermittelt, Informationen können jedoch in den Vorgangs-spezifischen Speicherbereichen weitergereicht werden. Das Programm TAC2 wählt für die Weitergabe von Informationen an TAC3 einen MPUT-Aufruf und beendet sich mit PEND PA. Es wird also kein Sicherungspunkt gesetzt. Im Programm TAC3 wird der Vorgang mit PEND FI beendet.

Asynchron-Vorgänge, die ihrerseits Aufträge absetzen

Während eines Asynchron-Vorgangs können wiederum Asynchron-Aufträge erzeugt werden. Dies können Ausgabe-Aufträge oder weitere Hintergrund-Aufträge sein. Bei verteilter Verarbeitung können aus einem Asynchron-Vorgang auch Dialog-Aufträge an Partner-Anwendungen gestellt werden, d.h. der Asynchron-Vorgang kommuniziert mit fernen Dialog-Vorgängen.

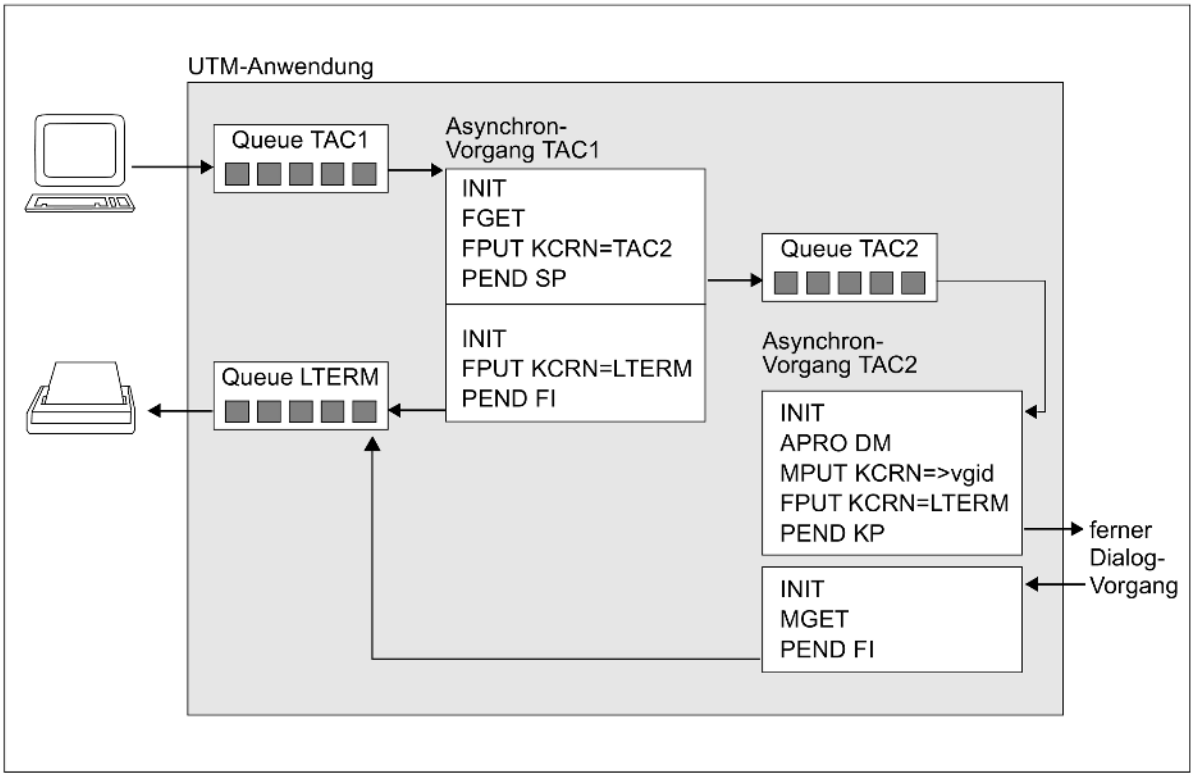


Bild: Asynchron-Vorgänge, die ihrerseits Aufträge absetzen

Im ersten Teilprogramm des Asynchron-Vorgangs TAC 1 wird ein Auftrag an den Asynchron-Vorgang TAC2 abgesetzt. Hierzu muss beim FPUT im Feld KCRN der Transaktionscode TAC2 angegeben werden. Da das Teilprogramm mit einem Sicherungspunkt endet, wird der Auftrag bereits zum Ende des Teilprogramms in die entsprechende Queue eingereiht.

Auch im Vorgang TAC2 wird im ersten Teilprogramm ein Asynchron-Auftrag abgesetzt, ein Ausgabe-Auftrag an den Drucker. Beim FPUT muss hierfür das Feld KCRN mit dem LTERM-Namen des Druckers versorgt werden. Da das Teilprogramm nicht mit einem Sicherungspunkt endet, wird der Auftrag erst zum Vorgangsende (nächster Sicherungspunkt) in die Queue des LTERMs eingereiht.

Der Vorgang TAC 2 ist in zwei Verarbeitungsschritte gegliedert. Das erste Teilprogramm sendet mit MPUT eine Nachricht an einen fernen Dialog-Vorgang. Das zweite Teilprogramm wird gestartet, wenn die Antwort vom fernen Vorgang eintrifft. Diese wird mit MGET gelesen.

Auftrags-Komplexe

Zusammen mit dem eigentlichen Asynchron-Auftrag ("Basisauftrag") können bis zu zwei **Quittungsaufträge** formuliert werden, die an das positive bzw. negative Ergebnis der Auftragsdurchführung gebunden sind. Sie werden ausgeführt, wenn der Basisauftrag abgewickelt ist. Mit den Quittungsaufträgen hat der Auftraggeber die Möglichkeit, auf ein positives oder negatives Auftragsergebnis zu reagieren. Ein Quittungsauftrag, der nicht zur Wirkung kommt - z.B. der negative Quittungsauftrag bei einem positiven Ergebnis - verfällt. Basisauftrag und Quittungsaufträge werden zusammen als **Auftrags-Komplex** bezeichnet.

Die folgende Tabelle zeigt einige exemplarische Ereignisse bei der Asynchron-Verarbeitung und die Wirkung dieser Ereignisse auf die Quittungsaufträge.

Beginn und Ende eines Auftrags-Komplexes legt man jeweils mit einem eigenen KDCS-Aufruf fest, dem Aufruf **MCOM BC** (begin of complex) und **MCOM EC** (end of complex). Beim MCOM BC-Aufruf definieren Sie die Komplex-Identifikation (Komplex-Id), das Ziel des Asynchron-Auftrags sowie die TACs der Asynchron-Programme, welche die positive bzw. negative Quittung bearbeiten sollen. Alle im Komplex erzeugten Aufträge werden mit DPUT-Aufrufen beschrieben, wobei als Ziel die Komplex-ID angegeben werden muss.

Ein Auftrags-Komplex kann sowohl in Dialog- als auch in Asynchron-Programmen definiert werden.

Ist der Basisauftrag eines Auftrags-Komplexes an einen fernen Asynchron-Vorgang gerichtet, dann muss die Vorgangs-Id vor Beginn des Auftrags-Komplexes vergeben werden (per APRO AM). Bei MCOM BC gibt man in KCRN die Vorgangs-Id an; die Quittungsaufträge müssen von der lokalen Anwendung bearbeitet werden.

Ereignis	**Wirkung auf Quittungsauftrag**
PEND FI im lokalen Asynchron-Vorgang	Start positiver Quittungsauftrag und Löschen negativer Quittungsauftrag
erfolgreiche Übertragung bei Hintergrund-Auftrag an fernen Asynchron-Vorgang	
positive Abdruckquittung vom Drucker oder von der Druckadministration	
Verarbeitung einer Nachricht einer TAC-Queue und Abschluss der Transaktion	
PEND ER/FR im lokalen Asynchron-Vorgang ohne Redelivery	Start negativer Quittungsauftrag und Löschen positiver Quittungsauftrag
Fehler beim Aufbereiten der Ausgabenachricht durch VTSU-B auf BS2000-Systemen	
Fehler bei Formatierung der Ausgabenachricht auf BS2000-Systemen	
Verarbeitung einer Nachricht einer TAC-Queue und Rücksetzen der Transaktion ohne Redelivery	
Zurückweisung des Auftrags bei Hintergrund-Auftrag an fernen Asynchron-Vorgang	
Löschen eines Auftrags per Administration	
Löschen des Ausgabeauftrags beim Verbindungsauf- oder Verbindungsabbau eines RESTART=NO Client.	

Ereignis	Wirkung auf Quittungsauftrag
PEND ER/FR im lokalen Asynchron-Vorgang mit Redelivery	ohne Wirkung, da der Basisauftrag noch erhalten bleibt
negative Abdruckquittung vom Drucker oder Zeitüberschreitung beim Warten auf eine Quittung	
Reihenfolge der Aufträge wird per Administration geändert	
Wiederholung eines Druckauftrags	
Verarbeitung einer Nachricht einer TAC-Queue und Rücksetzen der Transaktion mit Redelivery	
Löschen eines Auftrags mit Quittungsaufträgen per Druckadministration	Quittungsaufträge werden gelöscht; Protokollierung auf SYSLOG
KDCUPD übernimmt einen Auftrag nicht	Quittungsaufträge werden auch nicht übernommen (siehe KDCUPD-Protokoll)

Das folgende Bild veranschaulicht einen Auftrags-Komplex am Beispiel eines Druckauftrags.

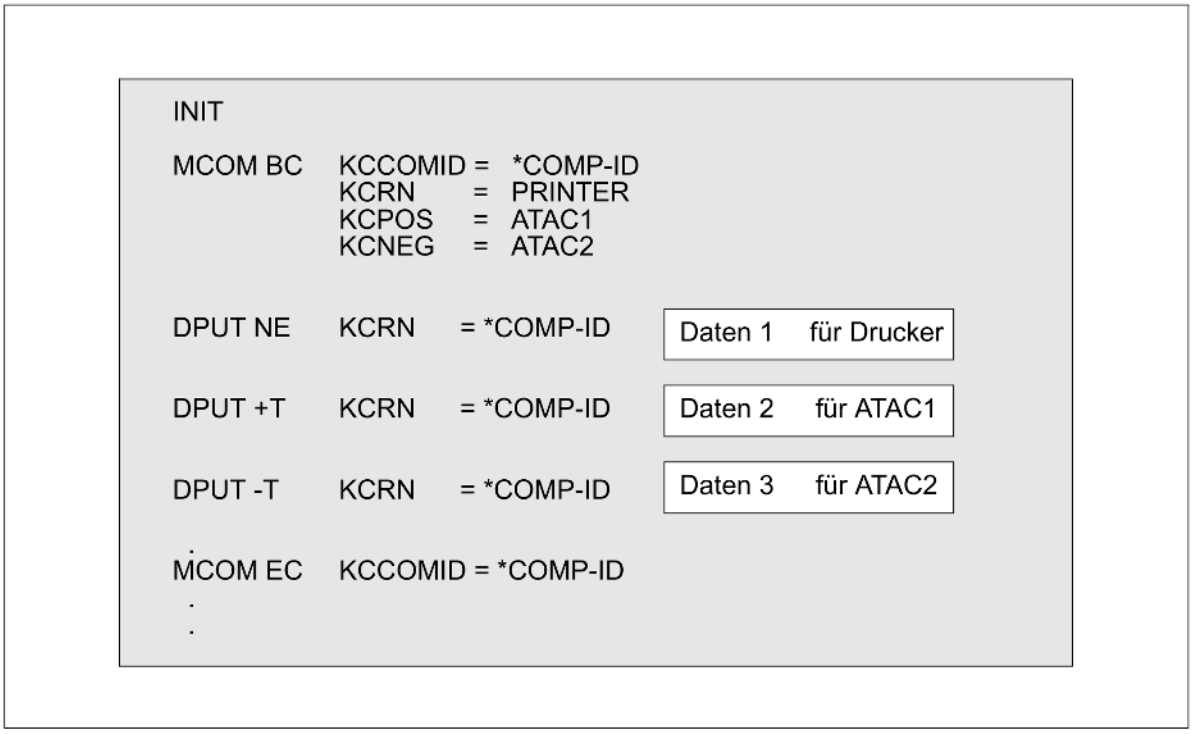


Bild: Auftrags-Komplex für einen Druckauftrag

2.4.2.5 Redelivery bei Hintergrundaufträgen

Wird ein Asynchron-Vorgang abnormal beendet durch PEND ER/FR oder System PEND ER, ohne dass zuvor eine Transaktion mit Commit abgeschlossen wurde, dann wird dieser Vorgang erneut gestartet und bekommt die FGET-Nachricht erneut zugestellt, vorausgesetzt, dies ist per Generierung eingeschaltet.

Ob die erneute Zustellung möglich ist und wie oft die erneute Zustellung wiederholt werden kann, wird per Generierung festgelegt (Operand REDELIVERY in der MAX-Anweisung). Beim FGET-Aufruf wird die Anzahl erneuter Zustellungen im KB-Rückgabebereich ausgegeben.

2.4.3 Nachrichten an Service-gesteuerte Queues

Bei Service-gesteuerten Queues wird die weitere Verarbeitung von Nachrichten, die ein Teilprogramm an eine Service-gesteuerte Queue sendet, allein von den Services der Anwendung bestimmt. openUTM sorgt nur für die Sicherung der Nachrichten in den Queues. Die Anwendungsprogramme müssen von sich aus die in diesen Queues gespeicherten Nachrichten lesen; sind in einer Queue keine Nachrichten vorhanden, so kann ein Anwendungsteilprogramm auch auf das Eintreffen einer Nachricht in einer Queue warten.

openUTM unterscheidet USER-Queues, TAC-Queues und Temporäre Queues. Die Eigenschaften dieser Queues werden in den Abschnitten „USER-Queues“ bis „Temporäre Queues“ beschrieben. Die zugehörigen MQ-Aufrufe sind in Abschnitt „MQ-Aufrufe der KDCS-Schnittstelle“ aufgelistet. Näheres zur Lebensdauer von Queues und Nachrichten in den Queues finden Sie in Abschnitt „Lebensdauer von Queues und Queue-Nachrichten“.

Service-gesteuerte Queues bieten neue Kommunikationsmöglichkeiten in einer Vielzahl von Fällen; so ist es unter anderem möglich, Message Queues für die Realisierung der folgenden Szenarien einzusetzen:

- zur Kommunikation voneinander unabhängiger Vorgänge in einer Anwendung
- zum Führen von "Pseudo-Dialogen" mit fernen Transportsystem-Anwendungen
- zur Parallelverarbeitung von Datenbankzugriffen in (Lese-)Transaktionen
- zum Senden von Nachrichten an UTM-Benutzer ("Mailbox-Funktionalität")
- zum Senden von Asynchron-Nachrichten an UPIC-Clients
- zum Senden von Nachrichten an Queues in anderen Anwendungen ("Remote-Queues", siehe Abschnitt „Service-gesteuerte Queues bei verteilter Verarbeitung“).
- zur Ausgabe von UTM-Meldungen an den UTM-Administrationsarbeitsplatz WinAdmin oder WebAdmin
- zur Realisierung einer anwendergesteuerten Verarbeitung von Asynchron-Nachrichten.
- zur Serialisierung von Teilprogrammen, die im Dialog und asynchron ablaufen
- zur Ausgabe von Daten fremder TLS-Blöcke an die Datenstation im Dialog-Vorgang
- als globale Speicherbereiche unbeschränkter Größe

Die Queues können mit dem Aufruf DADM administriert werden.

2.4.3.1 USER-Queues

USER-Queues sind permanente Service-gesteuerte Message Queues. Jedem generierten UTM-Benutzer steht jederzeit eine USER-Queue zur Verfügung. Auf USER-Queues kann im Prinzip jeder Service per Programmaufruf zugreifen, sofern er den Namen des Users kennt.

Mit Hilfe dieser Queue können zum Beispiel Nachrichten an den Benutzer am Terminal oder an einen über einen UPIC-Client angemeldeten Benutzer gesendet werden, wobei die USER-Queue als "Mailbox" eingesetzt wird. Die Queue kann außerdem zur Kommunikation zwischen dem Dialog-Vorgang des Benutzers und Asynchron-Vorgängen dienen, die unter derselben Benutzerkennung ablaufen.
Damit können Sie beispielsweise multiprozessorfähige, parallele Datenbankauskünfte, die viel Zeit kosten, auf mehrere Asynchron-Vorgänge verteilen, und die Ergebnisse im Dialog-Vorgang "aufsammeln".

Bei der Generierung kann für die USER-Queue in der KDCDEF-Anweisung USER ein Lese- und Schreibschutz vergeben werden (Parameter Q-READ-ACL bzw. Q-WRITE-ACL).
Einzelheiten zur Generierung eines Zugriffsschutzes finden Sie im openUTM-Handbuch „Anwendungen generieren“ unter der KDCDEF-Anweisung USER. Unabhängig von einem bestehenden Zugriffsschutz kann ein Benutzer in jedem Falle Nachrichten aus seiner eigenen USER-Queue lesen bzw. Nachrichten an diese schicken.

Zeitgesteuerte Nachrichten können nicht an USER-Queues gesendet werden.

Mit dem KDCS-Aufruf INIT PU kann ein Teilprogramm abfragen, wie viele Nachrichten für die Benutzerkennung vorliegen, unter der es abläuft.

2.4.3.2 TAC-Queues

TAC-Queues sind permanente Service-gesteuerte Message Queues. Jede TAC-Queue besitzt einen festen Namen, der mit der KDCDEF-Anweisung TAC ... TYPE=Q generiert wird. Auf TAC-Queues kann im Prinzip jeder Service per Programmaufruf zugreifen, sofern er den Namen der Queue kennt.

Mit Hilfe von TAC-Queues lassen sich zum Beispiel Remote Message Queues realisieren. Diese Remote Queues werden in der lokalen Anwendung dann über den LTAC-Namen angesprochen, siehe auch "Service-gesteuerte Queues bei verteilter Verarbeitung".

TAC-Queues können auch in Auftrags-Komplexen verwendet werden. Sowohl der Basisauftrag als auch die Quittungsaufträge können an TAC-Queues gerichtet werden.

Nachrichten können auch zeitgesteuert an TAC-Queues gesendet werden.

Bei der Generierung kann für die TAC-Queue in der KDCDEF- Anweisung TAC ein Lese- und Schreibschutz vergeben werden (Parameter READ-ACL bzw. WRITE-ACL). Mittels der dynamischen Administration (STATUS-Attribut) können TAC-Queues vollständig für schreibenden und/oder lesenden Zugriff gesperrt werden.

Einzelheiten zur Generierung von TAC-Queues finden Sie im openUTM-Handbuch „Anwendungen generieren“ unter der KDCDEF-Anweisung TAC.

2.4.3.3 Temporäre Queues

Temporäre Queues können dynamisch im Teilprogrammlauf durch den KDCS-Aufruf QCRE erzeugt und den Aufruf QREL gelöscht werden. Für diese Aufrufe ist keine Administrationsberechtigung nötig.

Temporäre Queues sind besonders zur Kommunikation zwischen Vorgängen geeignet: Ein Vorgang erzeugt eine Temporäre Queue und gibt den Namen dieser Queue den von ihm erzeugten Asynchron-Vorgängen mit. Später liest der Vorgang die Nachrichten, die diese Asynchron-Vorgänge an die Temporäre Queue schicken bzw. wartet auf diese Nachrichten. Anschließend wird die Temporäre Queue wieder gelöscht. Wurde eine Temporäre Queue neu erzeugt, dann können schon in derselben Transaktion Nachrichten in diese Queue geschrieben werden. Diese Nachrichten lassen sich jedoch erst nach dem erfolgreichen Abschluss der Transaktion lesen und administrieren.

Der Name der Temporären Queue kann entweder frei gewählt oder von openUTM automatisch erzeugt werden. Die automatische Erzeugung eines Namens durch openUTM hat den Vorteil, dass jeweils neue Namen vergeben werden und ein Name sich erst nach 100 Millionen Aufrufen wiederholt.

Eine Temporäre Queue kann jederzeit mit dem Aufruf QREL gelöscht werden. Bei erfolgreichem Transaktionsende werden alle Nachrichten der Queue gelöscht und Name sowie Tabellenplatz der Queue freigegeben. Alle Vorgänge, die auf Nachrichten der gelöschten Queue warten, werden fortgesetzt.

Zeitgesteuerte Nachrichten können nicht an Temporäre Queues gesendet werden.

Die maximale Anzahl der Temporären Queues, die erzeugt werden können, wird beim Generieren der Anwendung festgelegt (QUEUE-Anweisung, Operand NUMBER, siehe auch openUTM-Handbuch „Anwendungen generieren“). In der QUEUE-Anweisung kann mittels des Operanden QLEV (Queue-Level) auch die Anzahl der Nachrichten limitiert werden, die in der Queue gespeichert werden; damit können Sie die Pagepool-Auslastung begrenzen. Das Queue-Level kann aber auch im Aufruf QCRE dynamisch festgelegt werden. Die Angabe im Aufruf QCRE überschreibt eine etwaige Angabe aus der Generierung.

Ein Teilprogramm kann sich mit dem KDCADMI-Aufruf KC_GET_OBJECT (Objekttyp: KC_QUEUE) über die Namen aller existierenden Temporären Queues und deren Eigenschaften informieren.

2.4.3.4 MQ-Aufrufe der KDCS-Schnittstelle

Für Service-gesteuerte Queues bietet die KDCS die folgenden Aufrufe:

- DGET (**D**elayed free message **GET**)
  Der Aufruf DGET dient zum Lesen von Nachrichten oder -Teilnachrichten aus USER-, TAC- oder Temporären Queues sowie zum Warten auf eine Nachricht einer Service-gesteuerten Queue. Es ist Lesen mit anschließendem Löschen ("Verarbeiten") und Lesen ohne Löschen ("Browsen") möglich.
- FPUT (**F**ree message **PUT**)
  FPUT-Aufrufe dienen zum Senden von Nachrichten an TAC-Queues.
  Eine Nachricht kann auch aus mehreren Teilnachrichten bestehen. Für jede Teilnachricht ist dann ein eigener FPUT-Aufruf notwendig.
- DPUT (**D**elayed free message **PUT**)
  Mit dem DPUT-Aufruf wird eine Nachricht oder -Teilnachricht an eine USER-, TAC- oder Temporäre Queue gesendet. Zeitsteuerung sowie Basis- und Quittungsaufträge sind dabei nur für TAC-Queues möglich.
- QCRE (**Q**ueue **CRE**ate)
  Der Aufruf QCRE dient dazu, eine Temporäre Queue zu erzeugen.
- QREL ((**Q**ueue **REL**ease)
  Der Aufruf QREL dient dazu, eine Temporäre Queue zu löschen.
- DADM (**D**elayed free message **ADM**inistration)
  Mit DADM können Übersichtsinformationen über den gesamten Inhalt einer Queue oder gezielt über einzelne Elemente angefordert werden. Außerdem lässt sich mit DADM die Bearbeitungsreihenfolge steuern: Sie können Nachrichten vorziehen, einzelne Nachrichten oder den gesamten Inhalt der Queue löschen.

Das genaue Format dieser Aufrufe und weitere Informationen finden Sie im Kapitel „KDCS-Aufrufe“.

2.4.3.5 Lebensdauer von Queues und Queue-Nachrichten

Bei UTM-S bleiben alle Queues sowie alle Nachrichten in den Queues auch nach dem Anwendungsende erhalten. D.h. die Nachrichten in den Queues werden ausfallsicher gespeichert und sind nach einem darauf folgenden Neustart noch vorhanden.
Bei einer Neugenerierung können die Nachrichten mit dem Dienstprogramm KDCUPD übernommen werden.

Bei UTM-F bleiben bei einem Anwendungsende die USER- und TAC-Queues erhalten, die Temporären Queues gehen verloren. Die in diesen Queues gespeicherten Nachrichten gehen bei allen drei Arten von Queues verloren.

Redelivery von Queue-Nachrichten

Wurden Queue-Nachrichten verarbeitet und wird die Transaktion anschließend zurückgesetzt, dann werden die Nachrichten erneut in die Queue gestellt. Sie können mit DGET gelesen werden. Die maximale Anzahl erneuter Zustellungen kann per Generierung eingestellt werden (Operand REDELIVERY in der MAX-Anweisung). Die erneute Zustellung lässt sich mit diesem Operanden auch ausschalten.

Beim DGET-Aufruf wird die Anzahl erneuter Zustellungen im KB-Rückgabebereich ausgegeben.

Alternativ dazu können solche Nachrichten auch in der Dead Letter Queue gesichert werden.

2.4.3.6 Löschen von USER- und TAC-Queues mittels programmierter Administration

USER- und TAC-Queues können mittels programmierter Administration gelöscht werden.

Eine USER-Queue wird gelöscht, indem die zugehörige Benutzerkennung gelöscht wird. Die Löschung kann wahlweise sofort oder bei der nächsten Generierung wirksam werden. Das sofortige Löschen von USER-Queues ist jedoch nur für stand-alone-Anwendungen möglich.

Die Dead Letter Queue kann nicht gelöscht werden.

Die Löschung von TAC-Queues wird erst bei der nächsten Generierung wirksam.

2.4.3.7 Beispiele

Die folgenden Abschnitte zeigen an Hand typischer Anwendungsfälle, wie Sie Service-gesteuerte Queues einsetzen können und wie Sie die zugehörigen Teilprogramme erstellen müssen.

Beispiel 1: Kommunikation zweier Vorgänge in einer Anwendung

Das Bild veranschaulicht die Auswertung eines Hintergrundauftrages mit Hilfe einer Temporären Queue. Der Hauptvorgang "Kunde" in einem Call Center lässt die gewünschten Daten durch einen Hintergrundauftrag suchen und liest die Ergebnisse aus einer Temporären Queue bzw. wartet auf die entsprechende Nachricht, wenn sie zum Zeitpunkt des DGET noch nicht vorliegt.

UTM-Anwendung "Call Center"

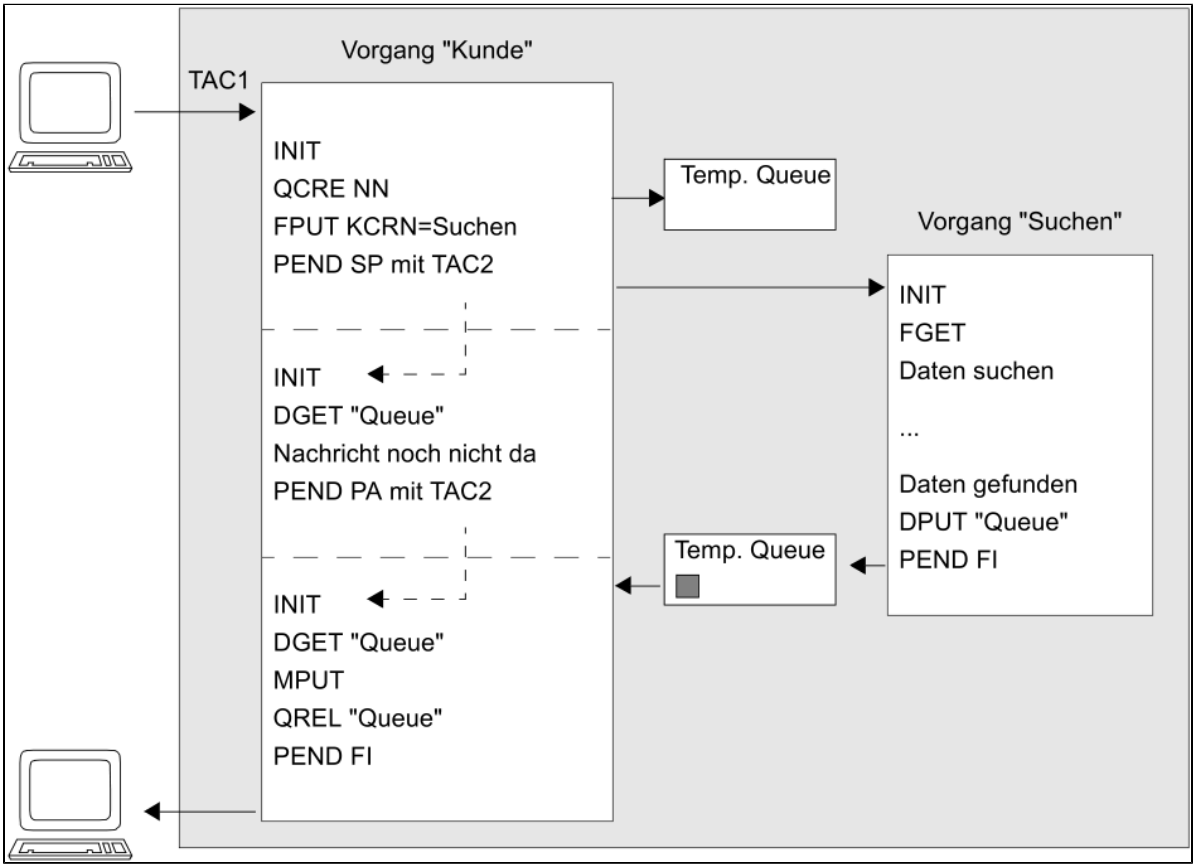


Bild: Kommunikation zweier unabhängiger Vorgänge mit Hilfe einer Temporären Queue

Mit QCRE NN wird eine Temporäre Queue erzeugt. Der Name der Queue wird in KCRQN zurückgegeben. Anschließend wird mit FPUT der Hintergrundauftrag "Suchen" erzeugt und die Transaktion sofort mit PEND SP beendet. Beim FPUT wird der Name der Queue im Nachrichtenbereich mitgegeben.

Der Vorgang "Suchen" liest die Nachricht mit dem Namen der Temporären Queue und sucht die gewünschten Daten. In der Zwischenzeit fragt der Hauptvorgang "Kunde" das Ergebnis der Hintergrundverarbeitung mit DGET ab. Da das Ergebnis in diesem Fall noch nicht vorliegt, beendet sich das aktuelle Teilprogramm des Hauptvorgangs; openUTM wartet auf das Eintreffen der Nachricht. Die Funktion ist so realisiert, dass in der Wartezeit für den Hauptvorgang kein Teilprogramm aktiv und kein Prozess gebunden ist.

Wenn der Hilfsvorgang "Suchen" die gewünschten Daten ermittelt hat, dann erzeugt er eine Nachricht für die Temporäre Queue und beendet sich, der DPUT-Auftrag wird damit ausgeführt.

Durch das Eintreffen der Nachricht wird der Hauptvorgang fortgesetzt. Dieser liest das Ergebnis aus der Temporären Queue und schickt es an den Client. Anschließend löscht er die Temporäre Queue und beendet sich (die Löschung wird erst beim erfolgreichen Transaktionsende wirksam).

Zur Steuerung der Abläufe in den Teilprogrammen muss ein Hilfsfeld `WaitForMsg` im Kommunikationsbereich KB angelegt werden, damit es von allen Teilprogrammen des Vorgangs zugreifbar ist. Kommt es wiederholt zum Returncode 08Z, so wird das überflüssige Warten durch einen RSET-Aufruf umgangen.

Teilprogramm TAC1

```
...
QCRE NN
FPUT mit KCRN=Suchen
kb.WaitForMsg = 0
PEND SP mit TAC2
```

Teilprogramm TAC2 zur Auswertung des Ergebnisses

```
...
DGET queue
if ( KCRCCC = "08Z" )
then if kb.WaitForMsg = 0
    then kb.WaitForMsg = 1
         PEND PA,<tac2>
    else RSET
else verarbeitung
QREL queue
...
PEND FI
```

Beispiel 2: "Pseudo -Dialoge" mit fernen Transportsystem-Anwendungen

Mit einer einfachen Modifikation von Beispiel 1 ergibt sich ein anderer Anwendungsfall: Ein Dialog- oder Asynchron-Vorgang möchte lesend auf eine Datenbank in einer anderen Anwendung zugreifen.

An die Stelle des Hilfsvorgangs in Beispiel 1 tritt ein Kommunikationspartner vom Typ APPLI oder SOCKET-USP. Dazu adressiert der FPUT-Aufruf im Hauptvorgang an Stelle eines Asynchron-TACs den LTERM-Namen der fernen Anwendung; von dieser Änderung abgesehen bleiben die Aufrufe und Abläufe in den beiden Vorgängen unverändert. Handelt es sich bei der fernen Anwendung um eine UTM-Anwendung, dann muss der TAC, der in der fernen Anwendung aufgerufen werden soll, am Anfang der FPUT-Nachricht stehen. Die ferne Anwendung muss ihre Antwortnachricht mit dem Namen der Queue beginnen, in der der Hauptvorgang die Antwort erwartet.

openUTM bestimmt das Ziel der Nachricht allein aus dem Namen der Queue. Daher darf der Name der Temporären Queue nicht mit einem TAC- oder LTERM-Namen der lokalen Anwendung übereinstimmen. Diese Namenskollision wird dadurch vermieden, dass der Name der Temporären Queue von openUTM vergeben wird (QCRE NN) und weder TAC- noch LTERM-Namen mit einer Ziffer beginnen.

Beispiel 3: Kommunikation mit mehr als einem Vorgang

Möchte ein Vorgang mit mehr als einem Hilfsvorgang gleichzeitig kommunizieren, dann ist der Ablauf ähnlich wie bei der Kommunikation mit nur einem Hilfsvorgang. Beim Lesen der Antworten muss jedoch darauf geachtet werden, dass alle Antworten gelesen werden; dazu muss möglicherweise mehrfach gewartet werden. Dabei müssen Sie darauf achten, dass auf jede Nachricht höchstens einmal gewartet wird.
Zur Steuerung der Abläufe in den Teilprogrammen werden in diesem Beispiel Hilfsfelder im KB angelegt, damit alle Teilprogramme des Vorgangs auf sie zugreifen können. Da es sein kann, dass nicht alle Antworten in einem Teilprogrammlauf gelesen werden können, wird mit Hilfe der Browse-Funktion eine Warteschleife realisiert. Die Antworten werden erst dann verarbeitet, nachdem die letzte Nachricht eingetroffen ist.

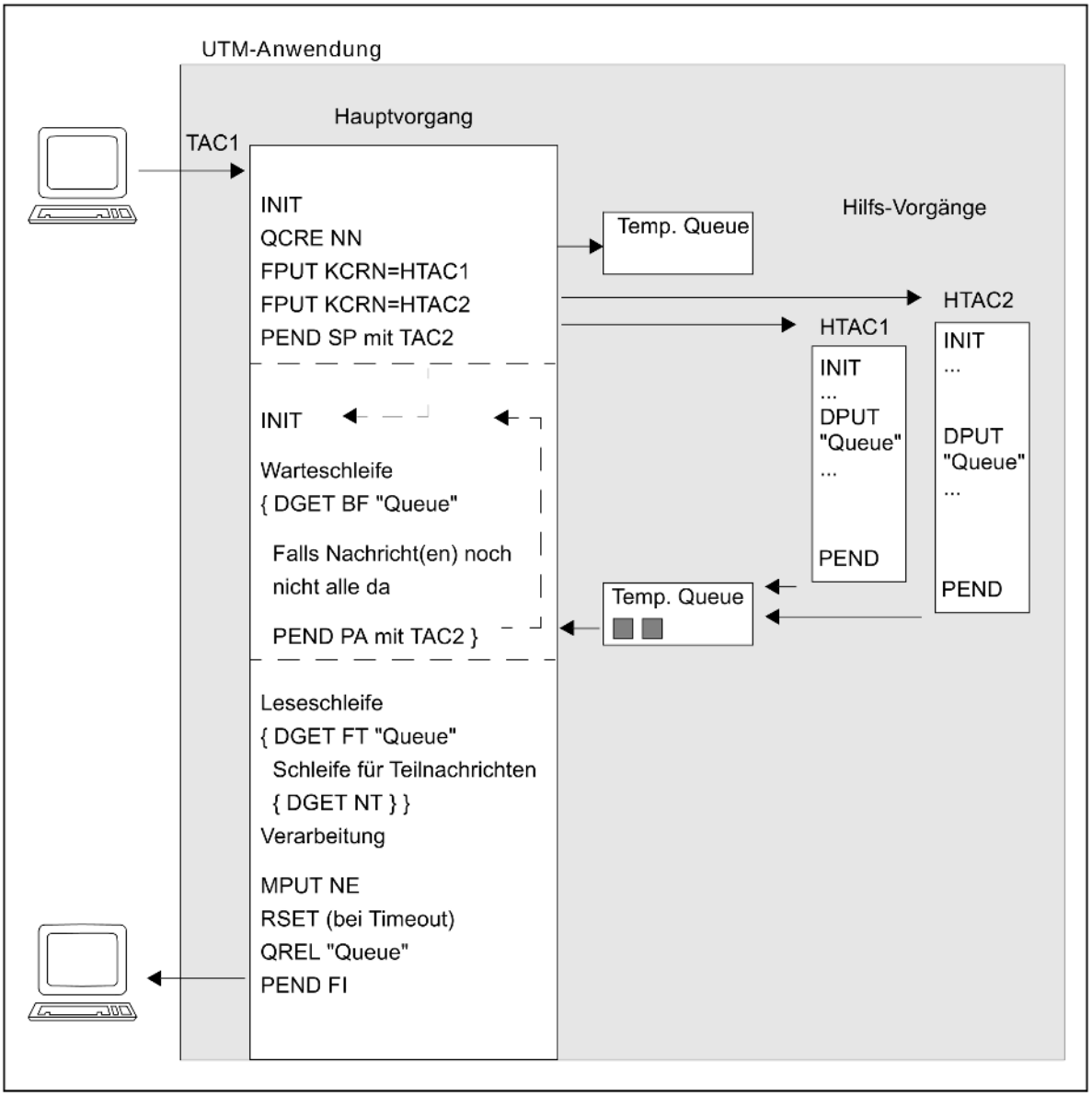


Bild: Kommunikation eines Hauptvorgangs mit zwei Hilfs-Vorgängen über Temporäre Queues

Für diese Kommunikation müssen Hilfsfelder verwendet sowie Schleifen programmiert werden. Die einzelnen Schritte werden nachfolgend in Tabellenform erläutert:

Programm TAC1	**Erläuterung**
INIT	
QCRE NN	Temporäre Queue erzeugen. Der Name der Queue wird in KCRQN zurückgegeben.
FPUT KCRN=HTAC1 FPUT KCRN=HTAC2	Nachrichten für die Hilfs-Vorgänge erzeugen. Der Name der Queue wird in der Nachricht mitgegeben.
kb.NrMsgs = 2 kb.WaitForMsg = 0 kb.i = 0 kb.kcgtm = spaces kb.kcdpid = spaces	Felder im KB-Programmbereich initialisieren. Diese Felder steuern den Ablauf in den Folgeprogrammen.
PEND SP KCRN=TAC2	Transaktion beenden, die Asynchron-Aufträge werden gestartet.
Programm TAC2	**Erläuterung**
INIT	
Timeout=0	Timout-Feld initialisieren.
for (kb.i<kb.NrMsgs AND Timeout=0) DGET BF mit Länge 0 kb.kcgtm kb.kcdpid	In einer Schleife wird versucht, mit Hilfe von DGET BF mit Warten alle Antworten abzuwarten, ohne die Daten zu lesen oder zu löschen, d.h. alle Nachrichten bleiben erhalten. Es muss darauf geachtet werden, dass auf jede Antwort höchstens einmal gewartet wird.
if (KCRCCC="08Z") then if kb.WaitForMsg=1 then Timeout=1 else kb.WaitForMsg=1 PEND PA KCRN=TAC2 else kb.i=kb.i +1 kb.WaitForMsg=0 kb.gtm=KCRGTM kb.kcdpid=KCRDPID	Muss gewartet werden, dann wird der Programmlauf mit PEND PA beendet; als Folge-TAC wird der aktuelle TAC angegeben. Ist eine Antwort eingetroffen, wird der Zähler erhöht. Die Erzeugungszeit und die DPUT-ID der Nachricht werden zwischengespeichert.

Programm TAC1	Erläuterung
if Timeout=0 then for (i<kb.NrMsgs) DGET FT for (KCRCCC ! = "10Z") DGET NT Verarbeitung MPUT NE	In zwei geschachtelten Schleifen werden alle Antworten vollständig gelesen, dann startet die Verarbeitung der Antworten. Das Ergebnis wird mit MPUT an den Client gesendet.
else RSET MPUT	Bei Timeout wird das überflüssige Warten durch einen RSET umgangen.
QREL "Queue" PEND FI	Vor Beenden des Vorgangs wird die Temporäre Queue wieder gelöscht.

Beispiel 4: Senden von asynchronen Nachrichten an UPIC-Clients

Asynchron-Nachrichten dürfen nicht an einen LTERM-Partner eines UPIC-Clients gerichtet sein; mit Hilfe von zwischengeschalteten Service-gesteuerten Queues können jedoch auch einem UPIC-Client asynchrone Ereignisse oder Meldungen zugestellt werden. Dazu richtet man die asynchronen Nachrichten entweder an die USER-Queue des UPIC-Clients oder man generiert für jeden UPIC-Client zusätzlich eine TAC-Queue. Der UPIC-Client startet nun parallel zu seinen normalen Dialog-Vorgängen einen weiteren Dialog-Vorgang, der nur die Aufgabe übernimmt, an dieser Queue zu warten bis eine Nachricht eintrifft, diese auszulesen und an den UPIC-Client weiterzuleiten.

Das folgende Diagramm zeigt die Realisierung mit Hilfe von USER-Queues:

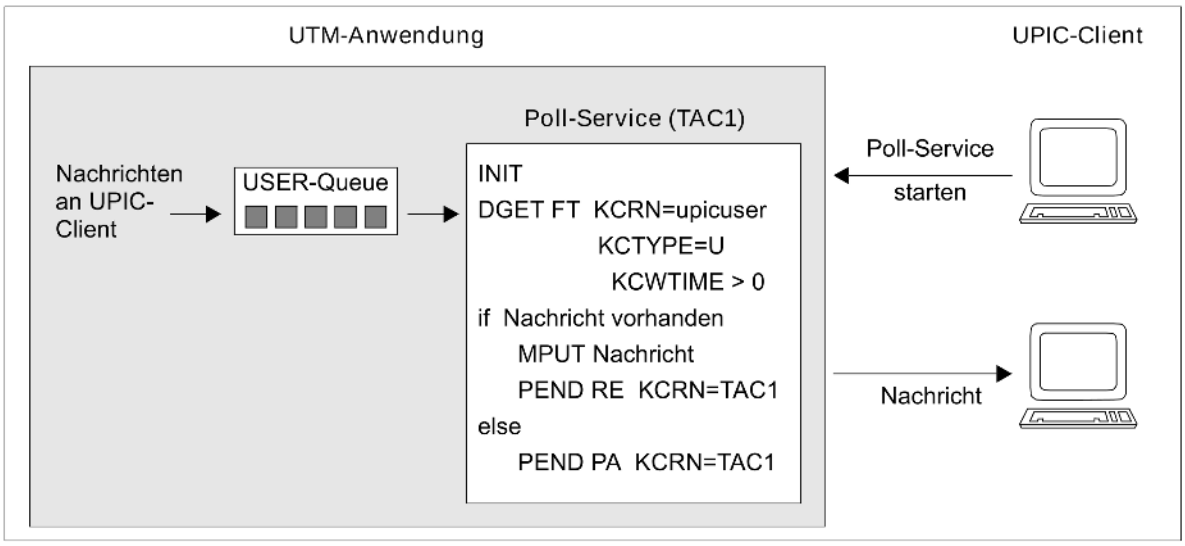


Bild: Senden asynchroner Nachrichten an UPIC-Clients mit Hilfe von USER-Queues

Beispiel 5: Serialisierung von Teilprogrammen

Soll ein Programm(abschnitt) serialisiert werden, der sowohl in Dialog- als auch in Asynchron-Vorgängen abläuft, so kann man eine TAC-Queue zur Serialisierung verwenden. TAC-Queues haben gegenüber GSSBs den Vorteil, dass außerhalb des Teilprogrammkontextes gewartet und der Prozess für andere Aufgaben frei wird.

Im folgenden Beispiel wird ein Abschnitt eines Teilprogramms, das im Dialog und asynchron ablaufen kann, mittels einer TAC-Queue gegen parallele Verarbeitung gesichert.

Generierung

```
TAC tacqueue,TYPE=Q
```

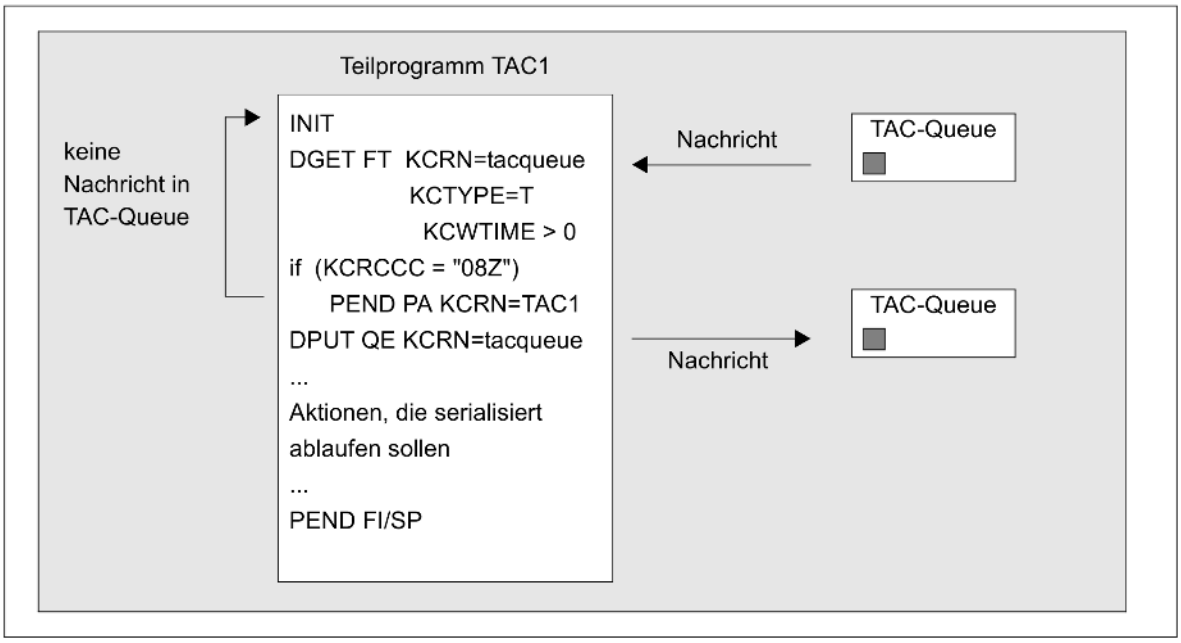


Bild: Serialisierung von Teilprogrammen mit Hilfe von TAC-Queues

Für den nächsten Vorgang, der die kritische Strecke durchlaufen soll, wird mit DPUT eine Nachricht an die TAC-Queue geschickt. Erst mit erfolgreichem Transaktionsende wird der DPUT-Aufruf wirksam.

Der hier beschriebene Mechanismus muss initialisiert werden, damit die kritische Teilprogrammstrecke durchlaufen werden kann. Dies lässt sich z.B. beim Anwendungsstart über ein MSGTAC-Programm realisieren, das auf eine Startmeldung (K050, K051) reagiert und eine Nachricht für die TAC-Queue erzeugt.

Beispiel 6: Ausgabe fremder TLS-Blöcke im Dialog-Vorgang

Zugriffe auf TLS-Blöcke einer fremden Datenstation sind nur in Asynchron-Vorgängen erlaubt. Das Ausgeben von Daten aus fremden TLS-Blöcken aus einem Dialog-Vorgang heraus an die Datenstation ist mit geringer Modifikation von Beispiel 1 möglich:

Der Dialog-Vorgang gibt dem Hilfsvorgang zusätzlich den Namen des TLS-Blocks und des LTERMs mit. Der Hilfsvorgang liest mit einem GTDA-Aufruf die gewünschten Daten und schickt diese an die Temporäre Queue.

Der Dialog-Vorgang liest die TLS-Daten aus der Temporären Queue und gibt sie an der Datenstation aus.

Beispiel 7: Service-gesteuerte Queues als globale Speicherbereiche

Der Zugriff auf anwendungsglobale Speicherbereiche (GSSBs, siehe "Globaler Sekundärer Speicherbereich (GSSB)") ist nur seriell möglich, d.h. während des Lesens/Schreibens ist ein solcher Bereich gesperrt. Wenn an Stelle eines GSSB eine Temporäre Queue mit der "Browse"-Funktion verwendet wird, dann können mehrere Vorgänge parallel auf den Speicherbereich zugreifen.

Außerdem haben Temporäre Queues gegenüber GSSBs weniger Beschränkungen. Der Speicherbereich einer Temporären Queue ist nicht eingeschränkt (GSSB: 32767 Byte) und es sind mehr Queues als GSSBs erlaubt (500.000 gegenüber 32767).

Mit Hilfe einer Temporären Queue lässt sich z.B. eine **Online-Auktion** realisieren:

- Mit DPUT QT/QE werden die Beschreibungen der Objekte nacheinander in eine Temporäre Queue gestellt. Für die Queue wird ein fester Name vergeben. Die Queue kann nachträglich erweitert werden.
- Mit DGET BF/BN können diese Beschreibungen durch Interessenten gelesen werden, auch parallel durch mehrere Vorgänge.
- Mit DGET PF/PN wird eine Beschreibung verarbeitet und damit aus der Queue genommen, z.B. nachdem ein Objekt versteigert wurde.
- Nach Ende der Auktion wird die Temporäre Queue mit QREL gelöscht.

Per Generierung lässt sich die Anzahl der Nachrichten in einer Temporären Queue beschränken (KDCDEF-Anweisung QUEUE, Operand QLEV). Falls nur die neuesten Nachrichten interessant sind, dann kann der Wrap-Modus gewählt werden (QUEUE, Operand QMODE=WRAP-AROUND). Sobald der Maximalwert erreicht ist, wird mit jeder neuen Nachricht die jeweils älteste Nachricht gelöscht.

2.5 Die KDCS-Speicherbereiche bei openUTM

openUTM bietet Teilprogrammen zum Schreiben und Lesen von Benutzerdaten verschiedene Speicherbereiche. Durch diese Speicherbereiche wird eine klare Trennung von Programm- und Datenbereichen gewährleistet und die Reentrant-Fähigkeit der Programme sichergestellt. Außerdem ermöglichen die Bereiche einen performanten und transaktionsgesicherten Austausch von Informationen zwischen Programmen und sorgen für effektiven Arbeitsspeichereinsatz. Einige Speicherbereiche sind speziell für statistische Zwecke und für die Protokollierung konzipiert.

Neben den hier dargestellten Speicherbereichen bietet Ihnen openUTM außerdem die Möglichkeit, Service-gesteuerte Queues für die Kommunikation zwischen Teilprogrammen zu verwenden, siehe Abschnitt „Nachrichten an Service-gesteuerte Queues“.

Folgende Speicherbereiche stehen zur Verfügung:

- Standard-Primärer Arbeitsbereich (SPAB)
- Kommunikations-Bereich (KB)
- Lokaler Sekundärer Speicherbereich (LSSB)
- Globaler Sekundärer Speicherbereich (GSSB)
- Terminal-spezifischer Langzeitspeicher (TLS)
- User-spezifischer Langzeitspeicher (ULS)
- Benutzer-Protokolldatei (nur zum Schreiben)
- AREA-Bereiche (AREA)

Die Anzahl der in einer UTM-Anwendung zur Verfügung stehenden GSSBs, LSSBs sowie TLS- und ULS-Blöcke wird bei der Generierung festgelegt; für die TLS- und ULS-Blöcke werden dabei gleichzeitig Namen vergeben; die Namen der GSSBs und LSSBs werden bei der Programmierung festgelegt.

Speicherort von Benutzerdaten

Die Benutzerdaten in KB, GSSBs, LSSBs, TLS- und ULS-Blöcken werden bei stand-alone-Anwendungen in der KDCFILE gespeichert.

Unix-, Linux- und Windows-Systeme
In einer UTM-Cluster-Anwendung besitzt jede Knoten-Anwendung eine eigene KDCFILE. Da es Knoten-lokale Daten und Cluster-weit gültige Daten gibt, ergeben sich im Vergleich zu einer stand-alone-Anwendung einige Besonderheiten:

- Knoten-lokale Daten werden ausschließlich in der KDCFILE der betreffenden Knoten-Anwendung gehalten. Knoten-lokale Daten sind z.B. TLS, Asynchron-Nachrichten, Hintergrund-Aufträge und Daten zu anderen knotengebundenen Vorgängen. Die KDCFILEs der einzelnen Knoten-Anwendungen sind zur Laufzeit nicht identisch.

- Cluster-weit gültige Daten werden in UTM-Cluster-Dateien wie z.B. Cluster-Pagepool, und Cluster-User-Datei gehalten, siehe openUTM-Handbuch „Konzepte und Funktionen“. Dazu gehören Anwenderdaten wie z.B. GSSB, ULS, die Vorgangsdaten von Benutzern sowie Passwörter.

Speicherkonzept

Das Speicherkonzept unterstützt folgende Funktionen:

- die Reentrant-Fähigkeit der Teilprogramme (SPAB)
- einen Zugriff auf die Daten einer Anwendung von mehreren parallel laufenden Vorgängen aus (GSSB, TLS, AREA)
- den Schutz benutzereigener Daten vor Zugriffen von fremden Benutzerkennungen aus (ULS)
- transaktionsgesichertes Ablegen von Protokollierungsdaten (Benutzer-Protokoll-Datei)
- die Unabhängigkeit parallel laufender Vorgänge (KB, LSSB)

openUTM stellt zwei Arten von Speicherbereichen bereit:

- Arbeitsspeicherbereiche (KB, SPAB)
  Dies sind Speicherbereiche, die den Teilprogrammen im Arbeitsspeicher zur Verfügung stehen. Sie sind direkt adressierbar.
- Sekundärspeicherbereiche (GSSB, LSSB, TLS, ULS, Protokoll-Dateien)
  Diese Speicherbereiche werden von openUTM auf Hintergrund-Speicher realisiert und mit speziellen KDCS-Aufrufen geschrieben und gelesen.

Je nach Funktion sind diese Bereiche zugeordnet:

- einem Teilprogramm (SPAB)
- einem Vorgang (KB und LSSB)
- einem LTERM-, LPAP- oder OSI-LPAP-Partner (TLS)
- einer Benutzerkennung (USER), Session (LSES) oder Association (OSI-LPAP ASSOCIATION NAMES)
- der Anwendung (Protokolldatei, AREA-Bereiche, GSSB)

Wird ein Vorgang in mehreren Verarbeitungsschritten oder in mehreren Teilprogrammen bearbeitet, so müssen evtl. Daten aufbewahrt und weitergereicht werden. Hierfür kann man verwenden:

- den Kommunikationsbereich (KB) und die
- Lokalen Sekundären Speicherbereich (LSSBs)

Den KB sollten Sie für Daten verwenden, die in jedem Verarbeitungsschritt benötigt werden.

LSSBs sollte man verwenden, wenn entweder die Daten nicht in jedem Teilprogrammlauf eines Vorgangs benötigt werden oder die Daten länger sind als 32 K und deswegen nicht in den KB passen.

Funktion und Lebensdauer der Speicherbereiche

Folgende Tabelle gibt eine Übersicht über Lebensdauer und Funktion der KDCS-Speicherbereiche. Das Bild auf der nächsten Seite zeigt darüber hinaus die speziellen KDCS-Aufrufe für die Speicherbearbeitung und verdeutlicht die Zuordnung der einzelnen Speicherbereiche.

Kurzname	Bereichstyp	Lebensdauer	Funktion
KB	Kommunikationsbereich	Start des Vorgangs bis Ende des Vorgangs	Zugriff auf aktuelle, von openUTM bereitgestellte Information; Datenaustausch zwischen Teilprogrammen eines Vorgangs
SPAB	Standard Primärer Arbeitsbereich	Start des Teilprogramms bis Ende des Teilprogramms	Parameterübergabe bei KDCS-Aufrufen; Nachrichtenpuffer
AREA	Per Generierung vereinbarter weiterer Speicherbereich	Dauer der Anwendung	Aufnahme von anwendungsglobalen Daten, die vorzugsweise nur gelesen werden. *Unix-, Linux- und Windows-Systeme* Bitte beachten Sie, dass AREAs im UTM-Cluster jeweils Knoten-spezifisch sind, d.h. ein Knoten kann auf die AREAs in einem anderen Knoten nicht zugreifen.
LSSB	Lokaler Sekundärer Arbeitsbereich	vom ersten Schreibaufruf bis zur expliziten Freigabe oder bis Vorgangsende	Datenaustausch zwischen Teilprogrammen eines Vorgangs
GSSB	Globaler Sekundärer Arbeitsbereich	vom ersten Schreibaufruf bis zur expliziten Freigabe oder bis zum Löschen der Anwendungsinformation	Austausch von Daten über Vorgangsgrenzen hinweg
ULS	User-spezifischer Langzeitspeicher	Generierung bis Änderung der Generierung	z.B. Statistiken spezifisch für bestimmte Benutzerkennungen (USER), LU6.1 Sessions (LSES) oder OSI-TP Associations (OSI-LPAP ASSOCIATION-NAMES)
TLS	Terminal-spezifischer Langzeitspeicher	Generierung bis Änderung der Generierung	Statistiken spezifisch für bestimmte Anschlusspunkte (LTERMs, LPAPs, OSI-LPAPs)
USLOG	Benutzer-Protokolldatei (User-Logging)	individuell festlegbar	Protokollierung

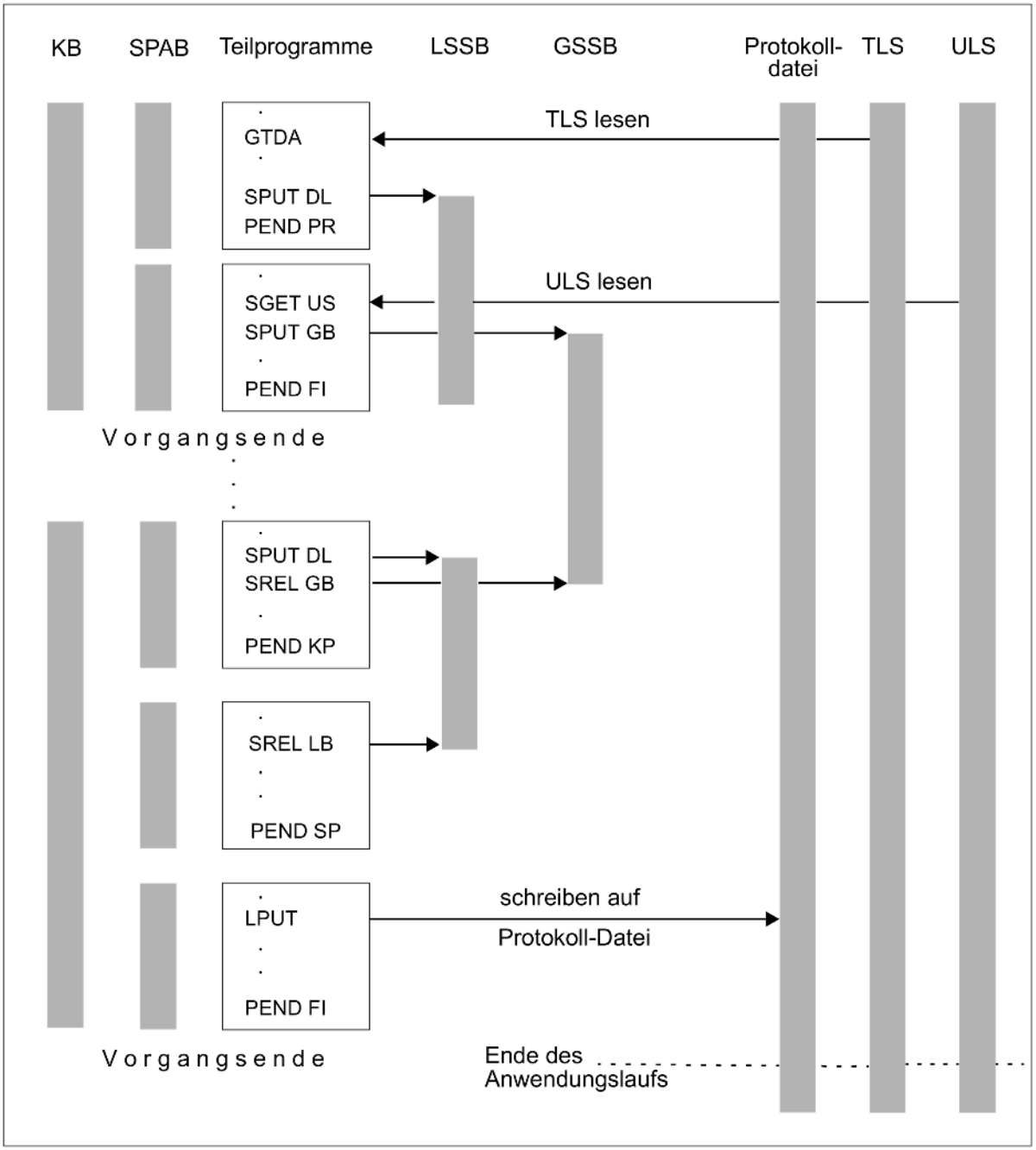


Bild: Zuordnung der KDCS-Speicherbereiche sowie KDCS-Aufrufe zur Speicherbearbeitung

2.5.1 Standard Primärer Arbeitsbereich (SPAB)

Jedem Teilprogramm ordnet openUTM standardmäßig einen SPAB zu, dessen maximale Länge bei der Generierung der Anwendung festgelegt wird (siehe openUTM-Handbuch „Anwendungen generieren“). Er steht dem Teilprogramm vom Programmstart bis zum PEND-Aufruf zur Verfügung.

In diesem Bereich können keine Daten über das Ende eines Teilprogrammlaufs hinaus aufbewahrt oder weitergegeben werden. Da der SPAB nur als Teilprogramm-spezifischer Arbeitsbereich dient, ist er **nicht** in die Transaktionssicherung einbezogen und wird von einem RSET-Aufruf nicht zurückgesetzt.

Im SPAB können untergebracht werden:

- der KDCS-Parameterbereich zur Ausführung eines Aufrufs

  Im Parameterbereich übergibt das Programm die Daten, die für einen KDCS-Aufruf notwendig sind. Die Einträge in die Felder des KDCS-Parameterbereichs hängen vom jeweiligen Aufruf ab (s. Kapitel „KDCS-Aufrufe“). Für die Strukturierung des Parameterbereichs stehen Ihnen Sprach-spezifische Datenstrukturen zur Verfügung (für COBOL im COPY-Element KCPAC und für C/C++ in der Include-Datei *kcmac.h)*.
- der Nachrichtenbereich (NB) für die Bereitstellung der Ein-/Ausgabe-Daten

  In den Nachrichtenbereich werden die Nachrichten eingetragen, die Sie mit den Aufrufen MGET, DGET, FGET, SGET, GTDA einlesen. Auch Informationen, die Sie mit Varianten der Aufrufe INIT, DADM, INFO und PADM anfordern, stellt openUTM im Nachrichtenbereich zur Verfügung. Bei der Ausgabe stellen Sie im Nachrichtenbereich die Daten bereit, die Sie mit MPUT, FPUT, DPUT, SPUT, PTDA bzw. LPUT übergeben.

  Die Adresse des Nachrichtenbereichs geben Sie im jeweiligen Aufruf an.
- weitere Bereiche mit Programmlauf-spezifischen variablen Daten.

Festlegbares Füllzeichen

Bei der Generierung einer UTM-Anwendung können Sie für diese Anwendung ein beliebiges Füllzeichen festlegen (im Operanden CLRCH der KDCDEF-Anweisung MAX). openUTM überschreibt dann den SPAB am Ende jedes Verarbeitungsschritts mit diesem Füllzeichen. Nach dem Start des Anwendungsprogramms ist dieser Bereich in der generierten Länge mit dem festgelegten Zeichen vorbesetzt.

Diese Funktion ist für den Programmtest wichtig, weil im Ein-Prozess-Betrieb einige Fehler so besser entdeckt werden können; darüber hinaus kann sie zu Datenschutzzwecken verwendet werden.

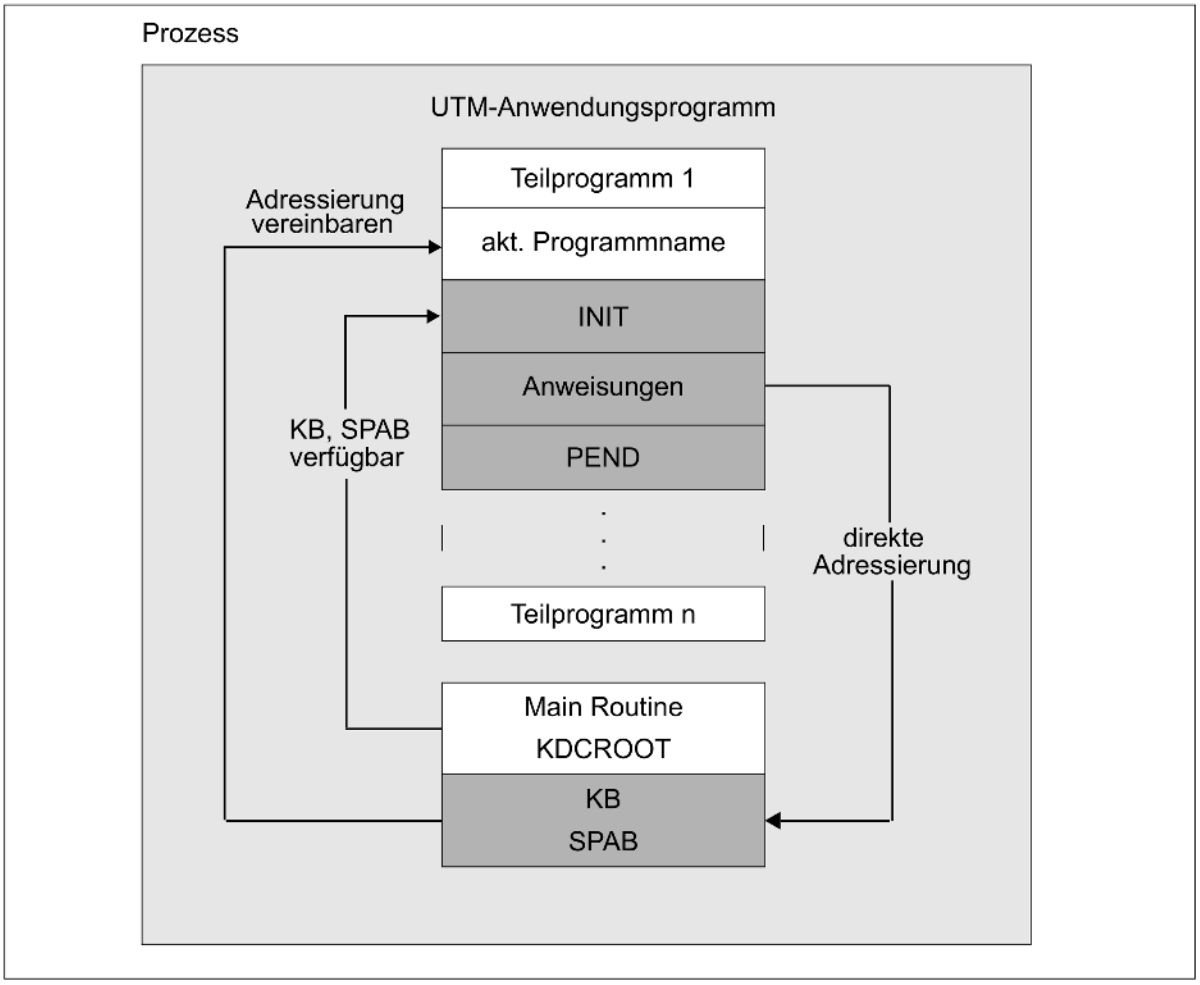


Bild: Benutzung der Primärspeicherbereiche (SPAB, KB)

2.5.2 Kommunikationsbereich (KB)

openUTM legt den KB an, wenn ein neuer Vorgang begonnen wird. Er bleibt solange erhalten, bis der Vorgang abgeschlossen ist. Sein Inhalt wird dem jeweils aktuell ausgeführten Programm übergeben. Der KB kann in seiner Größe den aktuell zu übergebenden Daten angepasst werden, d.h. er kann von Verarbeitungsschritt zu Verarbeitungsschritt wachsen oder auch wieder kleiner werden.

Am Anfang eines KB stehen zwei Bereiche fester Länge, die der Verständigung von openUTM mit dem Programm dienen:

- der KB-Kopf
- der KB-Rückgabebereich

Für die Strukturierung dieser Bereiche stehen Ihnen Sprach-spezifische Datenstrukturen zur Verfügung - für COBOL im COPY-Element KCKBC, für C/C++ in der Include-Datei *kcca.h*.

Im **KB-Kopf** finden Sie nach dem INIT-Aufruf und nach jedem weiteren Aufruf jeweils aktuelle Informationen über Vorgang, Teilprogramm und Kommunikationspartner.

Im **KB-Rückgabebereich** übergibt openUTM nach jedem Aufruf (außer PEND) seine Rückgabedaten an das Programm. Die Auswertung insbesondere des Returncodes gibt Aufschluss über die erfolgreiche oder gescheiterte Ausführung und kann darum zu entsprechenden Steuerungsmaßnahmen im Programm benutzt werden (siehe auch Abschnitt „Programmierung von Fehlerroutinen“).

Ferner können Sie im KB einen Bereich variabler Länge definieren, den **KB-Programmbereich**, dessen Struktur Sie frei festlegen können. Während der KB-Kopf und der KB-Rückgabebereich immer vorhanden sind, ist die Verwendung des KB-Programmbereichs wahlfrei. Er dient zur Weitergabe und Sicherung Vorgangs-spezifischer Daten beliebiger Art.

Die Maximallänge dieses Bereichs wird bei der Generierung der Anwendung festgelegt (Operand KB in der MAX-Anweisung). Im Programm geben Sie beim INIT die Länge an, die das Programm aktuell erwartet. Sie darf nicht größer sein als der von Ihnen generierte Wert des Operanden KB. Beim Erreichen eines Sicherungspunktes wird der KB-Programmbereich in der beim INIT angegebenen Länge gesichert. Geben Sie deshalb bei jedem INIT-Aufruf den KB nur in der unbedingt benötigten Mindestlänge an, um das Sichern nicht benötigter Informationen zu vermeiden.

Wenn mehrere Programme mit verschieden langen KB-Programmbereichen einen Vorgang bearbeiten, dann können bei der Datenübergabe von Programm zu Programm Längenkonflikte entstehen. Ein Programm erhält die Daten immer in der Länge, die das Vorgängerprogramm beim INIT im Feld KCLKBPRG angegeben hat. Stellt das empfangende Programm längere Datenfelder bereit, dann ist der Rest undefiniert. Definiert das empfangende Programm einen kürzeren KB, erhält es trotzdem die Daten in der vollen Länge. Die Verkürzung wirkt sich erst bei der erneuten Weitergabe der Daten aus (siehe Bild auf der folgenden Seite).

Festlegbares Füllzeichen

Bei der Generierung einer UTM-Anwendung können Sie für diese Anwendung ein beliebiges Füllzeichen festlegen (im Operanden CLRCH der KDCDEF-Anweisung MAX). openUTM überschreibt dann den KB am Ende des Vorgangs mit diesem Füllzeichen. Nach dem Start des Anwendungsprogramms ist dieser Bereich in der generierten Länge mit dem festgelegten Zeichen vorbesetzt.

Diese Funktion ist für den Programmtest wichtig, weil im Ein-Prozess-Betrieb einige Fehler so besser entdeckt werden können; darüber hinaus kann sie zu Datenschutzzwecken verwendet werden.

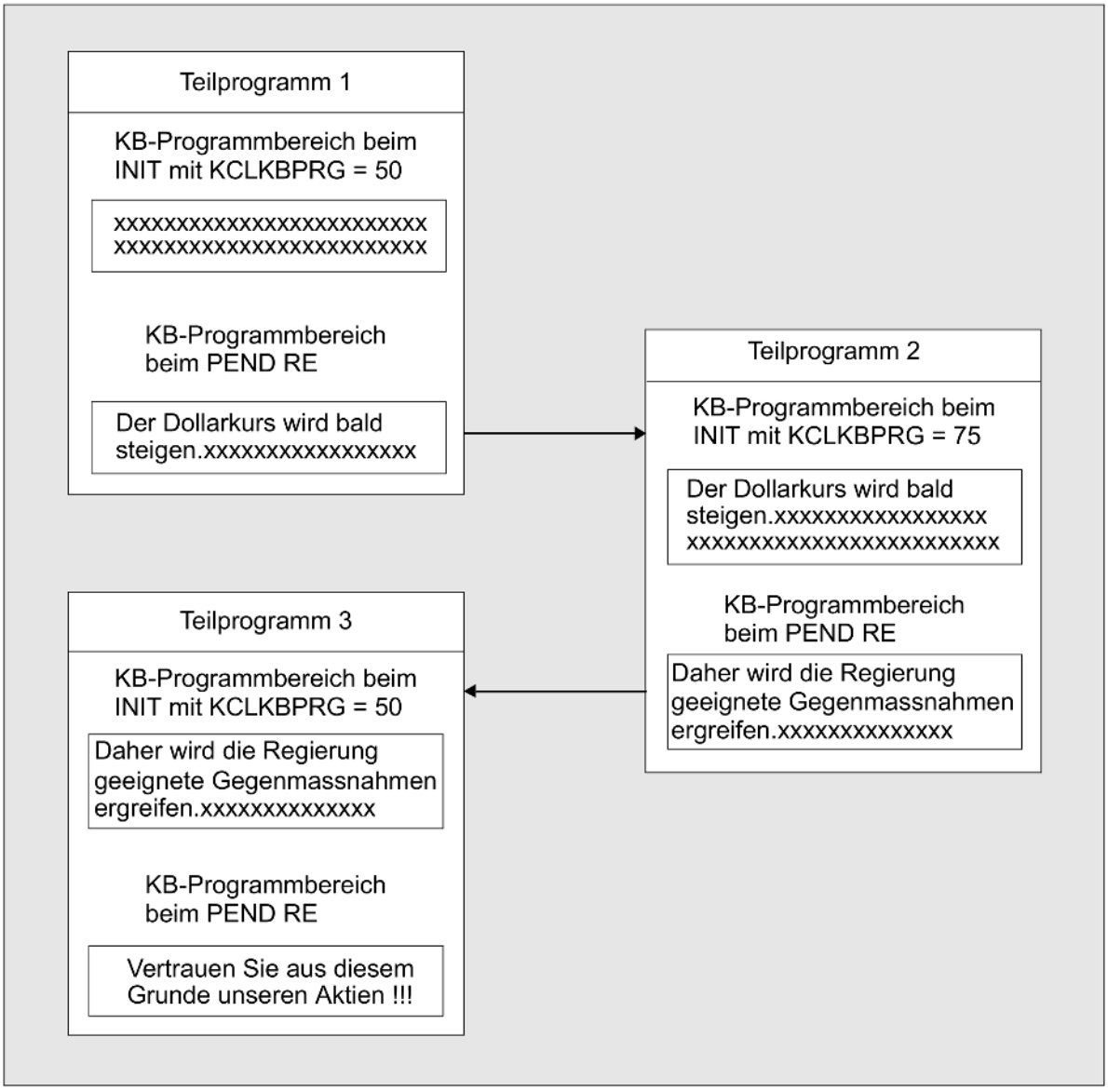


Bild: KB-Programmbereich für Datenübergabe

2.5.3 Lokaler Sekundärer Speicherbereich (LSSB)

LSSBs sind Vorgangs-spezifische Hintergrundspeicher, die zur Weitergabe von Daten zwischen Teilprogrammen innerhalb eines Vorgangs dienen. Ihre Inhalte werden in Standalone-Anwendungen in die KDCFILE ausgelagert, siehe „Speicherort von Benutzerdaten“ (Die KDCS-Speicherbereiche bei openUTM). Während openUTM den Kommunikationsbereich jedem Teilprogramm automatisch zur Verfügung stellt und danach sichert, wird auf den LSSB nur bei Bedarf zugegriffen. Er eignet sich deshalb besonders für Daten, auf die nur lesend zugegriffen wird, oder wenn zwischen Schreiben und Lesen der Daten mehrere Teilprogramme liegen, in denen die Daten nicht benötigt werden.

LSSBs können z.B. für folgende Fälle verwendet werden:

- Realisierung einer "Blätterfunktion" bei Dialog-Ausgaben
- "Temporäre" Datenerfassung

In einen LSSB können Sie mit einem SPUT-Aufruf Daten schreiben. Dabei geben Sie für den zu schreibenden Bereich einen Namen an. Dieser Name gilt nur für den lokalen Vorgang; verwenden zwei Vorgänge den gleichen Namen, dann bezeichnen sie damit zwei verschiedene LSSBs. Eingerichtet wird ein LSSB beim ersten schreibenden Zugriff innerhalb eines Vorgangs. Der Programmierer bestimmt dabei die Länge des LSSB. Ein LSSB kann maximal 32767 Bytes lang sein.

Lesen können Sie aus diesem Bereich mit dem SGET-Aufruf. Wird der Speicher nach dem SGET-Aufruf nicht mehr benötigt, kann er gleichzeitig freigegeben werden. Um einen LSSB freizugeben, können Sie auch den Aufruf SREL verwenden. Wenn der zugehörige Vorgang beendet ist, werden nicht freigegebene LSSBs automatisch freigegeben.

Die maximale Anzahl der LSSBs, die in einem Vorgang erzeugt werden können, wird beim Generieren der Anwendung festgelegt (MAX-Anweisung, Operand LSSB, siehe auch openUTM-Handbuch „Anwendungen generieren“).

Unix-, Linux- und Windows-Systeme
Die Inhalte von LSSBs werden in UTM-Cluster-Anwendungen in den Cluster-Pagepool ausgelagert, siehe „Speicherort von Benutzerdaten“ (Die KDCS-Speicherbereiche bei openUTM).

2.5.4 Globaler Sekundärer Speicherbereich (GSSB)

GSSBs sind ebenso wie LSSBs Hintergrundspeicher. Der GSSB unterscheidet sich vom LSSB durch seine Funktion, Lebensdauer und Verwendung. Der GSSB ermöglicht die Übergabe von Daten von einem Vorgang zu einem anderen. Die Datenübergabe ist bei UTM-S-Anwendungen auch über ein Anwendungsende hinaus möglich, d.h., die Daten stehen beim erneuten Start der Anwendung zur Verfügung.

Ein GSSB kann z.B. zur Datenübergabe zwischen Dialog- und Asynchron-Vorgängen verwendet werden, wenn Sie keine Service-gesteuerten Queues einsetzen, siehe "Nachrichten an Service-gesteuerte Queues".

Der GSSB steht allen Vorgängen offen. Daher müssen Sie für eine eindeutige Namensvergabe in allen Teilprogrammen der Anwendung sorgen. Von einem SPUT, SGET oder SREL-Aufruf bis zum Ende einer Transaktion ist ein GSSB für einen Vorgang reserviert. Damit werden Mehrfachzugriffe verhindert. Ein Aufruf aus einem anderen Vorgang auf den reservierten GSSB wartet, bis dieser wieder frei ist. openUTM sperrt jeweils den GSSB vom Zugriffszeitpunkt bis zum Ende einer Transaktion (siehe auch Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“).
Die maximale Wartezeit lässt sich bei der Generierung festlegen (KDCDEF-Anweisung MAX, Operand RESWAIT, Wert time1). Mit dem UNLK-Aufruf können Sie einen GSSB explizit entsperren, d.h. vorzeitig für einen eventuell wartenden Vorgang freigeben, sofern der GSSB nur gelesen wurde.

Ein GSSB (und auch sein Name) wird durch einen SPUT-Aufruf erzeugt.
Der Inhalt eines GSSB bleibt solange erhalten, bis er in einem Teilprogramm mit dem SREL-Aufruf freigegeben wird. Seine Lebensdauer ist nicht begrenzt. Auch nach einer Betriebsunterbrechung wird er für die UTM-Anwendung wieder bereitgestellt, wenn bei einer stand-alone-Anwendung die gleiche Datei KDCFILE wie vorher benutzt wird oder wenn die Benutzerdaten mit dem Tool KDCUPD in eine neue KDCFILE übertragen wurden.

Ein Teilprogramm kann sich mit dem KDCADMI-Aufruf KC_GET_OBJECT (Objekttyp: KC_GSSB) über die Namen aller existierenden GSSBs informieren.

Die maximale Anzahl der GSSBs, die erzeugt werden können, wird beim Generieren der Anwendung festgelegt (MAX-Anweisung, Operand GSSBS, siehe auch openUTM-Handbuch „Anwendungen generieren“).

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen werden GSSBs Cluster-weit unterstützt. D.h. sie werden im Cluster-Pagepool gespeichert und stehen somit für alle Benutzer in allen Knoten-Anwendungen zur Verfügung. Dadurch haben alle Knoten-Anwendungen dieselbe Sicht auf die Daten des Speicherbereichs, siehe auch „Speicherort von Benutzerdaten“ (Die KDCS-Speicherbereiche bei openUTM).

Der Inhalt eines GSSB bleibt solange erhalten, bis er in einem Teilprogramm mit dem SREL-Aufruf freigegeben wird. Seine Lebensdauer ist nicht begrenzt. Auch nach einer Betriebsunterbrechung wird er für die UTM-Anwendung wieder bereitgestellt, wenn bei einer UTM-Cluster-Anwendung der gleiche Cluster-Pagepool wie vorher benutzt wird oder wenn die Benutzerdaten mit dem Tool KDCUPD in eine neue KDCFILE bzw. den neuen Cluster-Pagepool übertragen wurden.

2.5.5 Terminal-spezifischer Langzeitspeicher (TLS)

Der TLS ist einem Anschlusspunkt (LTERM-, LPAP- oder OSI-LPAP-Partner) zugeordnet und dient zur Aufnahme von Informationen, die unabhängig von der Dauer eines Vorgangs und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Dieser Langzeitspeicher kann aus mehreren Blöcken bestehen, deren Namen mit TLS-Anweisungen definiert werden. Die definierten Block-Namen sind für alle LTERM- /LPAP- /OSI-LPAP-Partner identisch. Ein spezieller Block wird durch den Block-Namen und den Namen des LTERM- /LPAP- oder OSI-LPAP-Partners identifiziert (siehe openUTM-Handbuch „Anwendungen generieren“).

Ein TLS kann z.B. verwendet werden, um

- eine Statistik für einen LTERM- /LPAP- oder OSI-LPAP-Partner zu erstellen, oder um
- versuchte Zugriffsschutz-Verletzungen zu protokollieren, siehe Beispiel eines MSGTAC-Exits in Kapitel „Event-Funktionen“, "Asynchron-Vorgang MSGTAC".

Ein Teilprogrammlauf eines Dialog-Vorgangs kann nur auf Blöcke des "eigenen" TLS zugreifen, d.h. auf Blöcke des LTERM- /LPAP- oder OSI-LPAP-Partners, über den der Vorgang gestartet wurde.

Ein Teilprogrammlauf eines Asynchron-Vorgangs kann auf die Blöcke jedes LTERM- /LPAP- oder OSI-LPAP-Partners der UTM-Anwendung zugreifen.

Zum Lesen dient der Aufruf GTDA, zum Schreiben der Aufruf PTDA. openUTM sperrt jeweils den TLS-Block vom Zugriffszeitpunkt bis zum Ende einer Transaktion. Die maximale Wartezeit lässt sich bei der Generierung festlegen (KDCDEF-Anweisung MAX, Operand RESWAIT, Wert time1). Mit dem UNLK-Aufruf können Sie einen TLS jedoch explizit entsperren, falls er nur gelesen wurde (siehe auch Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“).

Der TLS dient zur Aufnahme von Informationen, die unabhängig von der Lebensdauer der Vorgänge und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Auch nach einer Betriebsunterbrechung wird er für die UTM-Anwendung wieder bereitgestellt, wenn die gleiche KDCFILE wie vorher benutzt wird oder wenn die Benutzerdaten mit dem Tool KDCUPD in eine neue KDCFILE übertragen wurden.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen werden TLS nur Knoten-lokal unterstützt, d.h. jede Knoten-Anwendung hat eigene TLS.

2.5.6 User-spezifischer Langzeitspeicher (ULS)

Der ULS ist ein Langzeitspeicher, der jeder Benutzerkennung (USER), LU6.1 Session (LSES) und OSI-TP Association (ASSOCIATION-NAMES der Anweisung OSI-LPAP) zugeordnet wird. Dieser Langzeitspeicher kann aus mehreren Blöcken bestehen, deren Namen mit ULS -Anweisungen definiert werden. Dabei wird pro ULS-Anweisung ein ULS-Block-Name definiert. Die definierten Block-Namen sind für alle Benutzerkennungen identisch. Ein spezieller Block wird durch den Block-Namen und den Namen der jeweiligen Benutzerkennung identifiziert (siehe openUTM-Handbuch „Anwendungen generieren“).

Der ULS kann z.B. verwendet werden, um Statistiken für Benutzerkennungen zu führen.

Ohne Administrationsberechtigung kann ein Teilprogramm nur auf die ULS-Blöcke der Benutzerkennung zugreifen, unter der es gestartet wurde. Für den Zugriff auf ULS-Blöcke fremder Benutzerkennungen ist Administrationsberechtigung notwendig.

Zum Lesen dient der Aufruf SGET, zum Schreiben der Aufruf SPUT. openUTM sperrt jeweils den ULS-Block vom Zugriffszeitpunkt bis zum Ende einer Transaktion. Die maximale Wartezeit lässt sich bei der Generierung festlegen (KDCDEF-Anweisung MAX, Operand RESWAIT, Wert time1). Mit dem Aufruf UNLK kann der ULS-Block jedoch explizit entsperrt werden, falls er nur gelesen wurde (siehe auch Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“).

Der ULS dient zur Aufnahme von Informationen, die unabhängig von der Lebensdauer der Vorgänge und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Auch nach einer Betriebsunterbrechung wird er für die UTM-Anwendung wieder bereitgestellt, wenn bei einer stand-alone-Anwendung die gleiche Datei KDCFILE wie vorher benutzt wird oder wenn die Benutzerdaten mit dem Tool KDCUPD in eine neue KDCFILE übertragen wurden.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen werden User-spezifische Langzeitspeicher Cluster-weit unterstützt. D.h sie werden im Cluster-Pagepool gespeichert und stehen somit für alle Benutzer in allen Knoten-Anwendungen zur Verfügung. Dadurch haben alle Knoten-Anwendungen dieselbe Sicht auf die Daten des Speicherbereichs, siehe auch „Speicherort von Benutzerdaten“ (Die KDCS-Speicherbereiche bei openUTM).

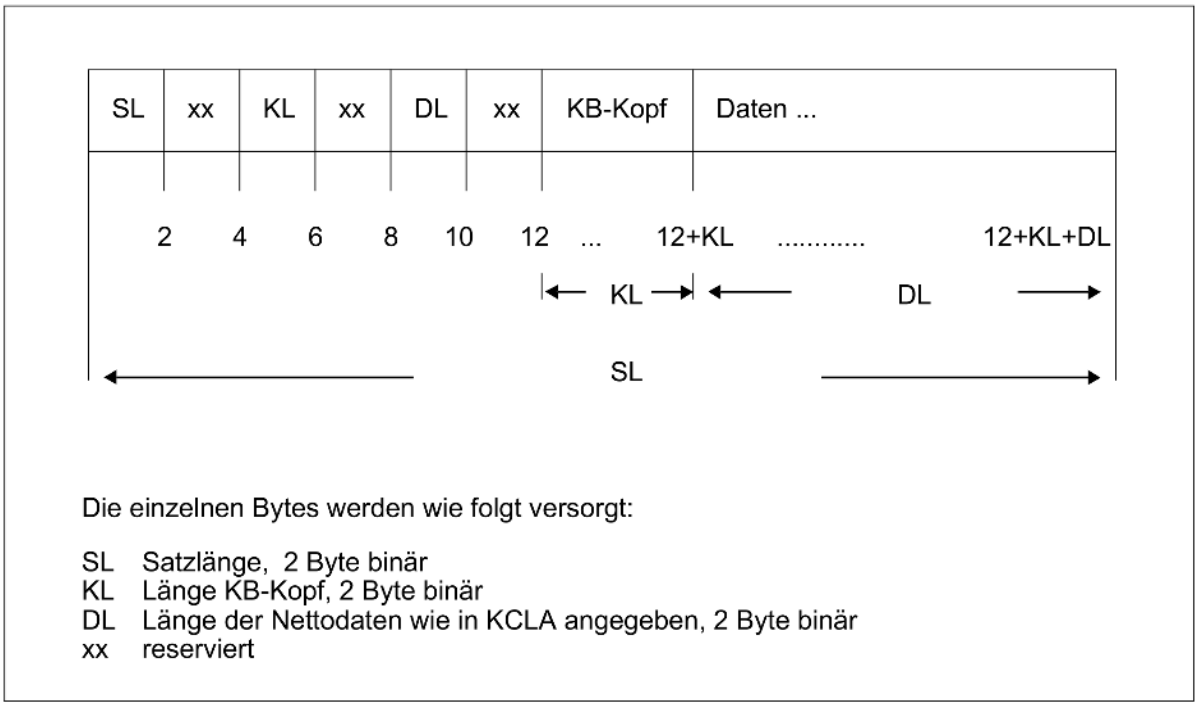
Der ULS dient zur Aufnahme von Informationen, die unabhängig von der Lebensdauer der Vorgänge und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Auch nach einer Betriebsunterbrechung wird er für die UTM-Anwendung wieder bereitgestellt, wenn bei einer UTM-Cluster-Anwendung der gleiche Cluster-Pagepool wie vorher benutzt wird oder wenn die Benutzerdaten mit dem Tool KDCUPD in den neuen Cluster-Pagepool übertragen wurden.

2.5.7 Benutzer-Protokolldatei

Die Benutzer-Protokolldatei (USLOG-Datei) dient zum Protokollieren von Benutzer-spezifischen Daten.

Unix-, Linux- und Windows-Systeme
In einer UTM-Cluster-Anwendung werden Benutzer-Protokolldateien nur Knoten-lokal unterstützt, d.h. jede Knoten-Anwendung schreibt in ihre eigene USLOG-Datei.

Die Benutzer-Protokolldatei hat folgenden Satzaufbau:



Die einzelnen Bytes werden wie folgt versorgt:

SL Satzlänge, 2 Byte binär
KL Länge KB-Kopf, 2 Byte binär
DL Länge der Nettodaten wie in KCLA angegeben, 2 Byte binär
xx reserviert

Bild: Satzaufbau der Benutzer-Protokolldatei

Jeder Datensatz wird mit einem LPUT-Aufruf erzeugt. Dabei versorgt openUTM den Vorspann des Datensatzes zusätzlich mit dem KB-Kopf. Der Inhalt von KB-Kopf entspricht dem Stand nach dem INIT-Aufruf. Sie müssen nur den Datenbereich bereitstellen mit den Daten, die Sie protokollieren wollen.

Näheres zur Benutzer-Protokolldatei finden Sie im openUTM-Handbuch „Einsatz von UTM-Anwendungen“ für die jeweilige Plattform.

2.5.8 Weitere Bereiche

Ein Teilprogramm kann bis zu 99 weitere Bereiche verwenden, die mit der KDCDEF-Anweisung AREA vereinbart werden (siehe openUTM-Handbuch „Anwendungen generieren“). AREA-Bereiche stehen der UTM-Anwendung als gemeinsame Speicherbereiche zur Verfügung. Die Struktur dieser Bereiche ist von openUTM nicht vorgegeben, sondern frei definierbar.

Die Adressen solcher Speicherbereiche werden neben KB und SPAB als zusätzliche Parameter beim Programmstart übergeben. Diese Bereiche unterliegen **nicht** dem Transaktionskonzept: openUTM sichert diese Bereiche nicht, setzt sie bei einem RSET-Aufruf nicht zurück und setzt auch keine Sperren. Für die Verwaltung dieser Speicherbereiche ist also allein das Anwendungsprogramm verantwortlich.

Informationen, wie Sie einen solchen Bereich für die jeweilige Programmiersprache definieren und einsetzen, finden Sie für C/C++ auf "Weitere Datenbereiche (AREAs)", für COBOL auf "COBOL-Teilprogramm als Unterprogramm".

> Falls Teilprogramme, die AREAs verwenden, aus einer Anwendung in eine andere Anwendung übernommen werden sollen, kann die Verwendung von AREAs aufgrund ggf. unterschiedlicher Parameterleisten zu Problemen führen. Alternativen zum Einsatz von AREAs finden Sie für die Programmiersprache C/C++ ebenfalls im Kapitel "Weitere Datenbereiche (AREAs)" bzw. für COBOL im Kapitel "COBOL-Teilprogramm als Unterprogramm".
>
> *Unix-, Linux- und Windows-Systeme*
> AREAs in UTM-Cluster-Anwendungen sind jeweils Knoten-spezifisch, d.h. ein Knoten kann auf die AREAs in einem anderen Knoten nicht zugreifen.

2.5.9 Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)

Wenn mehrere UTM-Transaktionen konkurrierend auf GSSB-, TLS- oder ULS-Bereiche zugreifen wollen, übernimmt openUTM die Synchronisation dieser Zugriffe.

Jeder Zugriff (Lesen, Schreiben oder Löschen) auf einen TLS, ULS oder GSSB hat zur Folge, dass openUTM diesen Bereich für Zugriffe anderer Transaktionen sperrt, und zwar

- bis zum nächsten Sicherungspunkt (PGWT CM, PEND RE, FI, SP oder FC) oder
- bis zur nächsten Rücksetzoperation (PGWT RB, RSET, PEND RS oder PEND ER/FR) oder
- bis zu einer Freigabe mit dem UNLK-Aufruf, falls aus dem Bereich nur gelesen wurde.

Eine Transaktion, die auf den gesperrten Bereich zugreifen möchte, wird von openUTM so lange in einen Wartezustand versetzt, bis

- die Sperre aufgehoben wurde
- oder die maximale Wartezeit abgelaufen ist.
- oder die Transaktion, durch die der Bereich gesperrt ist, sich mit PEND KP oder PEND PA/PR mit TAC-Klassenwechsel oder Warten auf DGET-Nachricht oder mit PGWT KP/PR/CM in einen Wartezustand unbestimmter Länge versetzt.

Dabei bleibt der Prozess blockiert und kann während der Wartezeit keine anderen Aufgaben übernehmen.

Diese maximale Wartezeit wird festgelegt mit dem Wert time1 des Operanden RESWAIT der KDCDEF-Anweisung MAX (Standard: 60 Sekunden, siehe auch openUTM-Handbuch „Anwendungen generieren“). Wartet eine Transaktion länger als die mit RESWAIT (time1) festgelegte Zeit, so lehnt openUTM den Zugriffsversuch mit dem Returncode KCRCCC = 40Z und KCRCDC = K810 ab.

openUTM lehnt einen solchen Zugriffsversuch in folgenden Fällen **sofort** ab,

- wenn der Bereich durch eine andere Transaktion gesperrt ist, die sich gerade mit PEND KP oder PEND PA/PR mit TAC-Klassenwechsel oder Warten auf DGET-Nachricht oder mit PGWT KP/PR in einen Wartezustand unbestimmter Länge versetzt hat (mit KCRCCC = 40Z und KCRCDC = K810), oder
- wenn das Warten zu einem Deadlock führen würde, so weit GSSB-, TLS- und ULS-Bereiche betroffen sind (mit KCRCCC = 40Z und KCRCDC = K820).
  Unix-, Linux- und Windows-Systeme
  In UTM-Cluster-Anwendungen muss dazu die Deadlock-Erkennung explizit eingeschaltet werden (siehe auch KDCDEF-Anweisung CLUSTER Parameter DEADLOCK-PREVENTION im openUTM-Handbuch „Anwendungen generieren“).

Greift eine Transaktion auf einen GSSB, ULS oder TLS nur lesend zu, so ist zu empfehlen, diesen Bereich möglichst schnell wieder freizugeben - insbesondere vor Datenbank-Aufrufen, vor PEND KP oder PEND PA/PR mit TAC-Klassenwechsel oder Warten auf DGET-Nachricht oder vor PGWT KP/PR - um die Wartezeiten anderer Transaktionen zu verkürzen.

Beim Zugriff auf LSSBs können solche Sperren nicht auftreten, da auf LSSBs nicht Vorgangs-übergreifend zugegriffen werden kann.

2.6 Programmierung von Fehlerroutinen

Nach KDCS-Aufrufen gibt openUTM im Rückgabebereich des Kommunikationsbereichs Returncodes zurück: im Feld KCRCCC nach jedem Aufruf einen KDCS-Rückgabecode gemäß DIN 66 265 (kompatibler Returncode) sowie evtl. im Feld KCRCDC einen UTM-spezifischen Returncode (KCRCDC). Diese Returncodes geben Aufschluss darüber, ob und wie der KDCS-Aufruf ausgeführt werden konnte bzw. warum openUTM ihn nicht ausführen kann.

Welche Fehlercodes bei welchen KDCS-Aufrufen auftreten können, ist bei jedem Aufruf beschrieben. Eine Zusammenstellung aller Fehlercodes finden Sie im openUTM-Handbuch „Meldungen, Test und Diagnose“.

Wenn nach einem Aufruf das Teilprogramm die Steuerung zurückerhält, sollte es also überprüfen:

- ob der Aufruf fehlerfrei ausgeführt wurde; dies erkennt das Teilprogramm daran, dass der KDCS-Returncode (im Feld KCRCCC) den Wert "000" hat.
- welche Fehler ggf. aufgetreten sind entsprechend dem angegebenen Returncode.

Ferner sind ggf. Zusatzinformationen wie tatsächlich übertragene Längen (Feld KCRLM), verwendete Formatkennzeichen (Feld KCRMF/kcrfn) usw. für die zu ergreifenden Maßnahmen interessant.

Bei schwerwiegenden Fehler erhält das Teilprogramm die Steuerung nicht mehr zurück. openUTM setzt in solchen Fällen die Transaktion zurück, beendet den Vorgang und erstellt einen PEND ER Dump; der Fehlercode, der zum Vorgangs-Abbruch geführt hat, kann der UTM Diagarea des PEND ER-Dumps entnommen werden. Näheres zur Analyse eines PEND ER Dumps finden Sie im openUTM-Handbuch „Meldungen, Test und Diagnose“.

Auf Unix- und Linux-Systemen haben Sie die Möglichkeit, die Steuerung in gewissen Fällen zu behalten, indem Sie eine anwenderspezifische Fehlerbehandlung durchführen, siehe „Anwenderspezifische Fehlerbehandlung (Unix- und Linux- Systeme)".

> **i** Nach einem PEND-Aufruf erhält das Teilprogramm die Steuerung nicht zurück. Der Rückgabebereich lässt sich deshalb anschließend nicht mehr im Teilprogramm auswerten. Treten in der PEND-Behandlung Fehler auf, dann reagiert UTM wie im vorstehenden Absatz beschrieben.

Das Teilprogramm kann mit RSET, PGWT RB oder mit PEND RS/ER/FR die Transaktion zurücksetzen; bei PEND ER wird ein Dump angefordert.

Es kann auch sein, dass openUTM einen Fehler meldet, weil eine der beteiligten Komponenten (z.B. das Formatierungssystem) einen Fehler an openUTM gemeldet hat.

Für solche Fehler führt openUTM intern eine Fehlerbehandlung durch, ohne dass sich das Teilprogramm darum kümmern muss. Wenn möglich, korrigiert openUTM den Fehler, sonst wird der Vorgang mit PEND ER abgebrochen und der spezifische UTM-Fehlercode entsprechend versorgt.

Beispiele für eine Fehlerroutine finden Sie bei den Programmbeispielen, für C/C++ „Programmierbeispiele in C/C++”, für COBOL „Programmier-Beispiele in COBOL".

2.7 Teilnachrichten

Die KDCS-Schnittstelle ermöglicht es, Teilnachrichten zu verarbeiten. Auf diese Weise lassen sich Nachrichten aus einzelnen Teilen zusammenstellen, um sie als ganze Nachricht dem Kommunikationspartner zu übermitteln.

Die Verwendung von Teilnachrichten bietet folgende Vorteile:

- Nachrichtenbereiche müssen nur so groß wie die größte Teilnachricht sein.
- Nachrichtenteile können gesondert verarbeitet werden.
- Formate können aus Teilformaten zusammengesetzt sein.

Teilnachrichten werden mit den Aufrufen MPUT NT, FPUT NT, DPUT NT oder DPUT QT an openUTM übergeben.

Teilnachrichten können gerichtet werden an ein Terminal, ein Client-Programm, eine Queue, einen Drucker, an ein Folge-Teilprogramm, einen lokalen Asynchron-Vorgang oder - bei verteilter Verarbeitung - an einen fernen Partner.

In allen Fällen, in denen mehrere Teilnachrichten an ein Teilprogramm gerichtet sind, muss dieses jede Teilnachricht mit einem eigenen DGET, MGET bzw. FGET lesen.

Auf BS2000-Systemen gilt Folgendes: Falls Sie Formate einsetzen, können Sie bei Nachrichten an Terminals und Drucker formatierte Teilnachrichten senden. Bei Dialog-Nachrichten an Terminals müssen Sie dann Teilformate verwenden (siehe "Einsatz von Teilformaten (BS2000-Systeme)"). Bei formatierten Asynchron-Nachrichten an Terminals wird jede Teilnachricht als eigene Nachricht behandelt. Bei formatierten Asynchron-Nachrichten an Drucker werden die Teilnachrichten zu einer logischen Einheit zusammengefasst.

Nachrichten mit Formatkennzeichen können auch an UPIC-Clients sowie an LU6.1-Partner gesendet werden. Für diese Partner dienen Formate nicht der Nachrichtenaufbereitung sondern vielmehr zur Beschreibung der Struktur der Benutzerdaten. Bei formatierten Nachrichten an diese Partner wird von openUTM keine Formatierungsroutine aufgerufen, vielmehr wird der Formatname an den UPIC-Client bzw. den LU6.1-Partner übermittelt.

Hinweis zu nachfolgendem Bild

Wird der FPUT im ersten Teilprogramm (TAC1) als FPUT NT geschrieben, so führt er ebenfalls zu einer eigenen Asynchron-Nachricht, weil die Nachricht beim PEND PR von openUTM abgeschlossen wird.

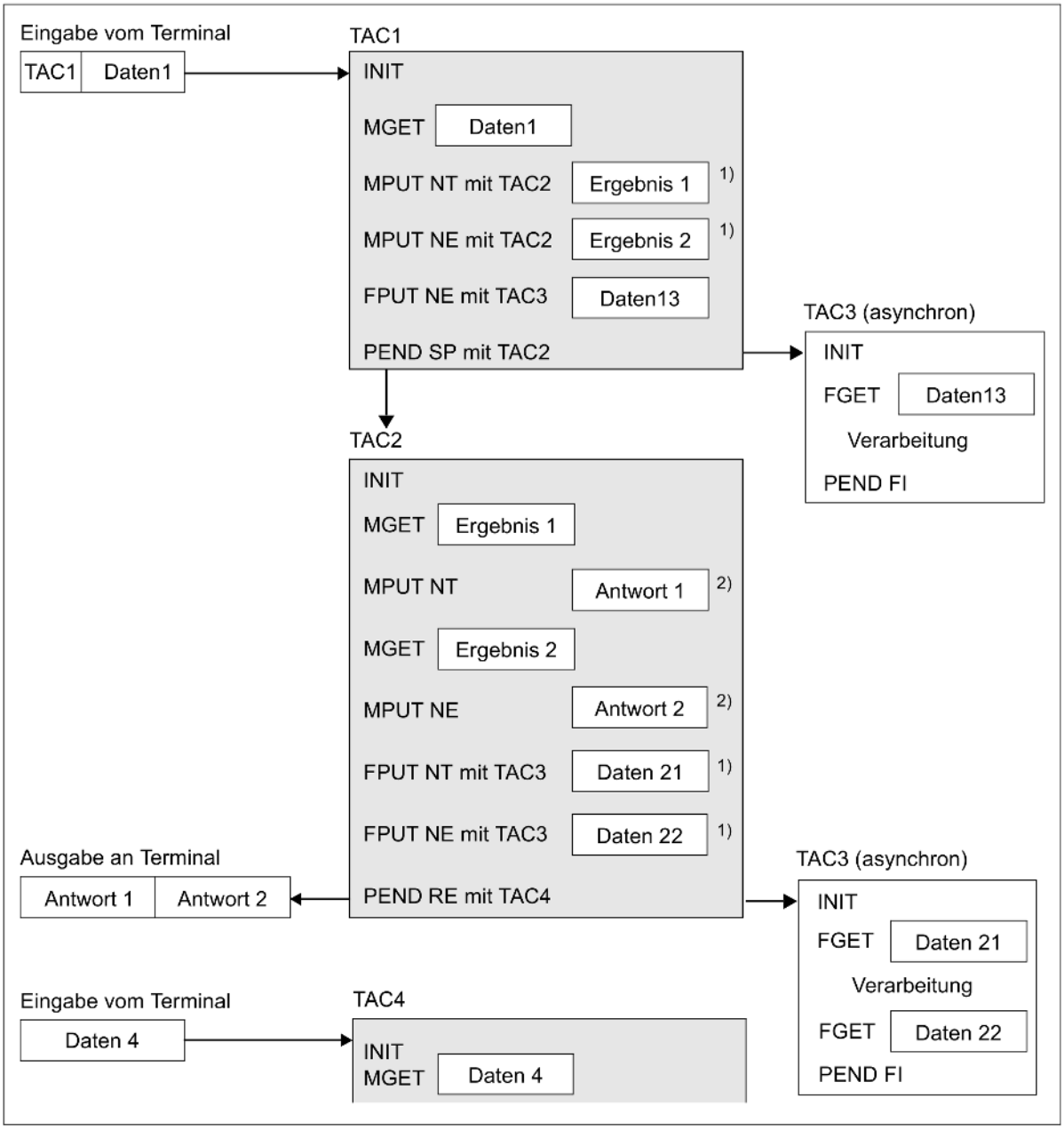


Bild: Teilnachrichten

[1)] Werden Teilnachrichten von Teilprogramm zu Teilprogramm gesendet, so muss jede Teilnachricht mit einem eigenen MGET NT oder FGET gelesen werden.

[2)] Bei Teilnachrichten an ein Terminal setzt openUTM diese zu einer Nachricht zusammen.

2.8 Kommunikationspartner einer UTM-Anwendung

Eine UTM-Anwendung kann mit verschiedensten Partnern kommunizieren. Kommunikationspartner einer UTM-Anwendung sind z.B.:

- Terminals

  Die KDCS-Schnittstelle ermöglicht den unmittelbaren Anschluss zeichenorientierten Terminals. Diese Terminals können Sie im Zeilenmodus einsetzen, oder in openUTM auf BS2000-Systemen auch Bildschirm-Formate (Masken) verwenden.

- UPIC-Clients

  UPIC-Clients sind Programme, die zur Kommunikation mit einer UTM-Anwendung die CPI-C-Schnittstelle mit UPIC als Trägersystem verwenden.

- Transportsystem(TS)-Anwendung

  Transportsystem-Anwendungen sind Programme, die zur Kommunikation mit einer UTM-Anwendung unmittelbar auf einer Transportsystem-Schnittstelle, wie z.B. Sockets, aufsetzen. TS-Anwendungen können auch andere UTM-Anwendungen sein.

- HTTP-Clients
  HTTP-Clients sind Programme, die zur Kommunikation mit einer UTM-Anwendung das HTTP-Protokoll verwenden und auf der Socket-Schnittstelle aufsetzen.

- Aus UTM-Vorgängen können Ausgabe-Aufträge am Drucker erzeugt werden (siehe Abschnitt „Ausgaben auf Drucker“). Auch Druckaufträge sind in das Transaktionskonzept einbezogen.

- OSI TP-, LU6.2- oder LU6.1-Partner

  Dabei kann es sich um eine andere UTM-Anwendung oder eine UTM-Client-Anwendung mit Trägersystem OpenCPIC handeln oder um Anwendungen, die auf Transaktionssystemen anderer Hersteller basieren.

Die Kommunikation mit allen diesen Partnern wird mit den gleichen KDCS-Aufrufen realisiert:

MPUT	zum Senden einer Dialog-Nachricht an den Partner
FPUT/DPUT	zum Senden einer Asynchron-Nachricht an den Partner oder eine Message Queue
MGET	zum Empfangen einer Dialog-Nachricht vom Partner
DGET	zum Lesen einer Asynchron-Nachricht aus einer Message Queue
FGET	zum Empfangen einer Asynchron-Nachricht vom Partner

Diese Aufrufe können auch zur Kommunikation innerhalb einer UTM-Anwendung eingesetzt werden:

- Mit MPUT kann eine Nachricht an ein Folge-Teilprogramm gesendet werden, die dieses dann mit MGET lesen kann. Für den Austausch zwischen den Teilprogrammen eines Vorgangs stehen jedoch auch Vorgangs-spezifische Speicherbereiche zur Verfügung (KB und LSSB siehe „Die KDCS-Speicherbereiche bei openUTM").
- Mit FPUT/DPUT lassen sich Asynchron-Nachrichten an lokale Asynchron-Vorgänge oder Message Queues senden, die von diesen mit FGET/DGET gelesen werden können.

2.9 Ausgaben auf Drucker

Druckerausgaben können Sie auf zwei verschiedene Arten realisieren:

- Hardcopy auf einen zentralen oder lokalen Drucker
- Ausgabe-Aufträge an Drucker (Druckaufträge), auch Spool-Betrieb genannt

Auf Windows-Systemen wird die transaktionsgesicherte Druckausgabe nicht unterstützt.

2.9.1 Hardcopy-Betrieb mit openUTM auf BS2000-Systemen

Automatischen Hardcopy-Betrieb realisieren Sie über die Bildschirmfunktion KCREPR im Feld KCDF des KDCS-Parameterbereichs. Bei #Formaten erfolgt die Steuerung per Globalattribut.

Zusätzlich ist es möglich, die Hardcopy-Funktion über Editprofile auszulösen (Angabe in KCMF/kcfn). Bei der Definition des Editprofils setzen Sie hierzu in der KDCDEF-Anweisung EDIT den Operanden HCOPY=Y (siehe openUTM-Handbuch „Anwendungen generieren“).

Beim Hardcopy-Betrieb wird die Nachricht mit MPUT, FPUT oder DPUT an das Terminal gesendet, der Ausdruck wird angestoßen. D.h. in KCRN werden beim MPUT Leerzeichen und beim FPUT/DPUT der LTERM-Name des Terminals eingetragen.

Der Terminal-Benutzer kann auch durch Drücken der Taste den aktuellen Bildschirminhalt auf dem zugeordneten ausgewählten Drucker ausgeben.

2.9.2 Druckaufträge

Druckaufträge werden mit den Message-Queuing-Aufrufen FPUT- oder DPUT erzeugt, siehe Abschnitt „Nachrichten an UTM-gesteuerte Queues“ und Kapitel „KDCS-Aufrufe“. Dabei geben Sie im Feld KCRN den LTERM-Namen des Druckers an. openUTM trägt den Auftrag in die entsprechende Queue ein.

Druckaufträge können Sie

- mit einem einzigen FPUT- oder DPUT-Aufruf erzeugen (NE im Feld KCOM) oder
- mit mehreren FPUT- oder DPUT-Aufrufen (für Teilnachrichten) aufbauen: NT im Feld KCOM, beim letzten FPUT- bzw. DPUT-Aufruf NE.

Teilnachrichten eines Druckauftrages behandelt openUTM bezüglich Reihenfolge und Fehlerbehandlung als Ganzes. Bei Druckerbündeln sendet openUTM zu einem Auftrag gehörende Teilnachrichten an **einen** Drucker. Ausgeben können Sie im Zeilenmodus.

Auf BS2000-Systemen können Sie Druckaufträge auch im Formatmodus ausgeben.

Auf BS2000-Systemen gibt es für RSO-Drucker zusätzlich die Möglichkeit, mit FPUT RP oder DPUT RP eine Parameterliste an RSO zu übergeben. RSO setzt die Parameter abhängig vom Druckertyp um.

Administration von Message Queues und Druckersteuerung

Die den LTERMs zugeordneten UTM-gesteuerten Message Queues, die die Druckaufträge enthalten, können Sie mit dem KDCS-Aufruf DADM administrieren (siehe "DADM Administration von Message Queues").

Der Aufruf DADM (delayed free message administration) bietet folgende Möglichkeiten:

- Informationen über Aufträge einer Message Queue in den Nachrichtenbereich lesen
- die Bearbeitungsreihenfolge von Aufträgen einer Message Queue ändern
- einzelne Aufträge oder auch eine gesamte Message Queue löschen
- Fehlerhafte Nachrichten aus der Dead Letter Queue verschieben

Für die Druckersteuerung steht Ihnen der KDCS-Aufruf PADM zur Verfügung. PADM (printer administration) bietet folgende Funktionen:

- Quittungsmodus für ein Druckersteuer-LTERM ein- oder ausschalten
- eine Druckausgabe bestätigen oder wiederholen
- die Zuordnung Drucker zu LTERM-Partner ändern
- den Druckerstatus ändern, d.h. Drucker sperren und wieder freigeben, Verbindung zu einem Drucker ab- oder aufbauen
- Informationen über einen Drucker in den Nachrichtenbereich lesen
- Informationen über zu quittierende Druckausgaben lesen

Ausführliche Informationen über die Administration von Message Queues, den Ablauf der Druckausgaben und die Möglichkeiten der Druckersteuerung finden Sie im openUTM-Handbuch „Anwendungen administrieren“.

Vermeiden von Engpass-Situationen bei hohem Druckaufkommen

Die Message Queues werden im Pagepool der KDCFILE gespeichert. Um Engpass-Situationen bei hohem Druckaufkommen zu vermeiden und einen Pagepool-Überlauf auszuschließen, können Sie im Teilprogramm und bei der Generierung folgende Vorkehrungen treffen.

- Im Teilprogramm:

  Den Returncode 40Z/K701 nach FPUT/DPUT-Aufrufen an Drucker auswerten bzw. die Meldung "`K041 Die Warnungsstufe # für PAGEPOOL wurde überschritten`" in einem MSGTAC-Teilprogramm behandeln und z.B.

  - Verbindungen zu den Druckern aufbauen, an die viele Nachrichten anstehen, damit die Nachrichten gedruckt werden und anschließend von openUTM gelöscht werden können
  - oder per DADM-Administration FPUTs löschen, oder TACs, die FPUTs an Drucker erzeugen, sperren.
- Per Generierung:
  - Mit QLEV= (LTERM-Anweisung) die Maximalanzahl der Nachrichten festlegen, die openUTM in der entsprechenden Message Queue zwischenspeichert. Druckaufträge werden erst beim Transaktionsende berücksichtigt. Weitere Nachrichten an diesen Drucker weist openUTM mit 40Z zurück.
  - Mit QAMSG= (LTERM-Anweisung) bestimmen Sie, ob die Nachrichten an einen nicht mit der Anwendung verbundenen Drucker zwischengespeichert werden.
  - Abschätzen, wieviel zusätzlicher Platz im Pagepool für Drucker-Nachrichten benötigt wird und im PGPOOL-Operanden der MAX-Anweisung einen hinreichend großen Wert angeben, Empfehlungen siehe openUTM-Handbuch „Anwendungen generieren“.

2.10 Unterstützung von Ausweislesern auf BS2000-Systemen

openUTM unterstützt Ausweisleser an Terminals.

Sie können den Ausweisleser auf zwei Arten verwenden:

- Zugangsschutz durch Ausweisprüfung beim Anmelden
- Dateneingabe per Ausweis

Sie können den Ausweisleser nicht gleichzeitig auf beide Arten verwenden: Nach Anmelden mit Ausweisleser kann der Terminal-Benutzer im Folgenden den Ausweis nicht zur Dateneingabe verwenden.

2.10.1 Anmelden an die Anwendung mit Ausweisleser (BS2000-Systeme)

Benutzerkennungen für eine UTM-Anwendung können so konfiguriert werden, dass der Zugang zu der Anwendung über eine Benutzerkennung nur mit einem speziellen Ausweis (Magnetstreifenkarte) möglich ist: entweder bei der Generierung mit der KDCDEF-Anweisung USER Operand CARD oder mittels dynamischer Konfiguration (KC_CREATE_OBJECT, Objekttyp KC_USER).

Meldet sich ein Benutzer über eine Benutzerkennung an, für die Ausweisprüfung konfiguriert ist, dann wird der Benutzer dazu aufgefordert, den Ausweis in den Leser zu stecken. openUTM prüft die Ausweisinformation. Stimmt sie mit dem überein, was in der Konfiguration für diese Benutzerkennung generiert wurde, so kann dieser mit der Anwendung arbeiten.
Steckt bei einer Dialog-Eingabe der Ausweis im Leser, dann besetzt openUTM das Feld KCAUSWEIS/kccard im KB-Kopf mit "A".

Der Benutzer darf den Ausweis nicht aus dem Leser nehmen, bevor er sich mit KDCOFF von der Anwendung abgemeldet hat, denn sonst bricht openUTM die Verbindung zum Terminal ab.

Die Ausweisinformation können Sie in den Teilprogrammen z.B. mit dem KDCS-Aufruf INFO, Operationsmodifikation "CD" (CARD) abrufen.

2.10.2 Dateneingabe per Ausweis (BS2000-Systeme)

Wird der Ausweisleser nicht zur Berechtigungsprüfung verwendet, so können auf BS2000-Systemen auf folgende Weise Daten über den Ausweisleser eingegeben werden:

Bei Ausgabe einer Dialog-Nachricht (MPUT) mit der Bildschirmfunktion KCCARD oder einem Editprofile mit SPECIN=I wird die Tastatur gesperrt und das Einlegen einer Magnetstreifenkarte in den Ausweisleser angefordert (das sollte aus dem Text der Dialog-Nachricht deutlich hervorgehen).

Das Folgeteilprogramm kann mit **MGET** die Daten der Magnetstreifenkarte wie eine normale Bildschirmeingabe lesen.

Wenn der Benutzer keine Magnetstreifenkarte zur Verfügung hat, kann er die Tastatur durch Drücken der Taste K14 (oder ESC zusammen mit :) entsperren. In diesem Fall erhält das Folgeteilprogramm beim MGET-Aufruf den Returncode, den Sie der Taste K14 bei der Generierung mit der KDCDEF-Anweisung **SFUNC** zugeordnet haben.

Wird die Magnetstreifenkarte aus dem Ausweisleser entnommen, so wiederholt openUTM die letzte Dialog-Ausgabe (interner KDCDISP).

Verfügbarkeit prüfen

- Die Verfügbarkeit des Ausweises können Sie im Feld KCAUSWEIS/kccard im KB-Kopf prüfen. Dort setzt openUTM das Kennzeichen "A", wenn bei der letzten Eingabe der Ausweis gesteckt hat.
- Die Verfügbarkeit eines Ausweislesers können Sie mit einem INFO CK-Aufruf überprüfen: Am Returncode des INFO CK-Aufrufes können Sie erkennen, ob an dem Terminal ein Ausweisleser vorhanden ist oder nicht. INFO CK rufen Sie sinnvollerweise *vor* einem MPUT-Aufruf mit KCCARD auf. Bei einem MPUT mit KCCARD an ein Terminal ohne Ausweisleser beendet UTM den Vorgang abnormal.

3 Zusammenarbeit mit Datenbanken

openUTM unterstützt die koordinierte Zusammenarbeit mit Datenbanksystemen. Die UTM-Transaktionen und die Transaktionen des Datenbanksystems werden dabei von openUTM über **Two-Phase-Commit** synchronisiert (siehe openUTM-Handbuch „Konzepte und Funktionen“).

Eine UTM-Anwendung kann mit mehreren Datenbanksystemen koordiniert zusammenarbeiten. Das bedeutet, dass **eine** UTM-Transaktion Aufrufe an **verschiedene** Datenbanksysteme enthalten darf.

Beendet ein UTM-Teilprogramm einen Dialog- oder Verarbeitungsschritt mit PEND KP, so wird die Transaktion im Normalfall in einem anderen Prozess fortgesetzt. Bei der Beendigung des Teilprogrammlaufs mit PEND PA/PR kann es ebenfalls sein, dass der Verarbeitungsschritt in einem anderen Prozess fortgesetzt wird. In verteilten OSI TP Transaktion erfolgt die Transaktionsbeendigung nach den KDCS-Aufrufen PEND SP, RE, FI oder PGWT CM im Normalfall ebenfalls in einem anderen Prozess. In einer solchen Umgebung müssen alle an der Transaktion beteiligten Datenbanksysteme den Prozesswechsel unterstützen.

Darüber hinaus ist es im Rahmen von verteilter Transaktionsverarbeitung möglich, Daten von mehreren unterschiedlichen Datenbanksystemen auf verschiedenen Rechnern innerhalb einer verteilten Transaktion zu bearbeiten.

Rücksetzen von Transaktionen

Im Fehlerfall gewährleistet openUTM, dass alle an der Transaktion beteiligten Datenbanken auf einen gemeinsamen Sicherungspunkt zurückgesetzt werden. Für den Programmierer entsteht keinerlei Aufwand für die Koordination zwischen openUTM und Datenbanksystemen.

Das Datenbanksystem kann von sich aus DB-Transaktionen zurücksetzen, z.B. um Langzeitsperren aufzulösen. Auch in diesem Fall werden die Transaktionen synchronisiert. Das Teilprogramm wird mit einem Rückkehrcode des entsprechenden DB-Systems informiert.

Interne Schnittstelle zwischen openUTM und Datenbanksystemen

openUTM steuert die Zusammenarbeit mit Datenbanksystemen über eine einheitliche und neutrale Schnittstelle und ist damit entkoppelt von den implementierungsabhängigen Spezifika der unterschiedlichen Datenbanksysteme.

Auf Unix-, Linux- und Windows-Systemen ist dies die von X/Open standardisierte XA-Schnittstelle, auf BS2000-Systemen wird neben der XA-Schnittstelle die funktionell vergleichbare Schnittstelle IUTMDB angeboten.

Unterstützte Datenbanksysteme

openUTM auf BS2000-Systemen unterstützt die Koordination mit folgenden Datenbanksystemen:

- UDS/SQL
- SESAM/SQL
- Oracle
- LEASY (das Dateisystem LEASY verhält sich gegenüber openUTM wie ein Datenbanksystem).

Auf Unix-, Linux- und Windows-Plattformen unterstützt openUTM die Koordination mit dem Datenbanksystemen Oracle.

Kopplung von openUTM mit Datenbanksystemen

Mit welchen Datenbanksystemen eine UTM-Anwendung koordiniert zusammenarbeiten soll, legen Sie beim Generieren der UTM-Anwendung mit einer KDCDEF-Anweisung fest: Für Unix-, Linux- und Windows-Systeme ist dies die Anweisung RMXA, für BS2000-Systeme die Anweisung DATABASE (siehe openUTM-Handbuch „Anwendungen generieren“).

Hinweis für BS2000-Systeme
Einige Datenbanksysteme bieten dem Anwendungsprogramm zum Aufruf verschiedene Schnittstellen, die als CALL-Schnittstelle oder als Sprachelemente der Programmiersprache (z.B. COBOL) realisiert sind. Welche Schnittstellen die Teilprogramme der UTM-Anwendung für die Kommunikation mit der Datenbank nutzen sollen, bestimmen Sie in der KDCDEF-Anweisung DATABASE.

Hinweis für Unix-, Linux- und Windows-Systeme sowie für XA-fähige Datenbanken auf BS2000-Systemen
Bei der Generierung einer XA-fähigen Datenbank wird keine Schnittstelle für das Anwendungsprogramm festgelegt. Die Schnittstelle hängt vom jeweiligen Resource-Manager ab.

Hinweis für die Nutzung von XA
Es gibt in der Regel einen statischen und einen dynamischen XA Switch. Eine Datenbank kann eine oder auch beide Varianten anbieten. Wenn die Datenbank einen dynamischen XA Switch anbietet, sollten Sie diesen nutzen. Dadurch wird die Ressourcenbelegung im Datenbanksystem minimiert.

3.1 UTM-Transaktion und DB-Transaktionen

Ein Vorgang ist in eine oder mehrere UTM-Transaktionen strukturiert. Diese Strukturierung wird von den Teilprogrammen des Vorgangs bestimmt. Eine DB-Transaktion wird von den Teilprogrammen bei Bedarf eröffnet, also wenn lesende oder ändernde Zugriffe auf die Datenbanken notwendig sind.

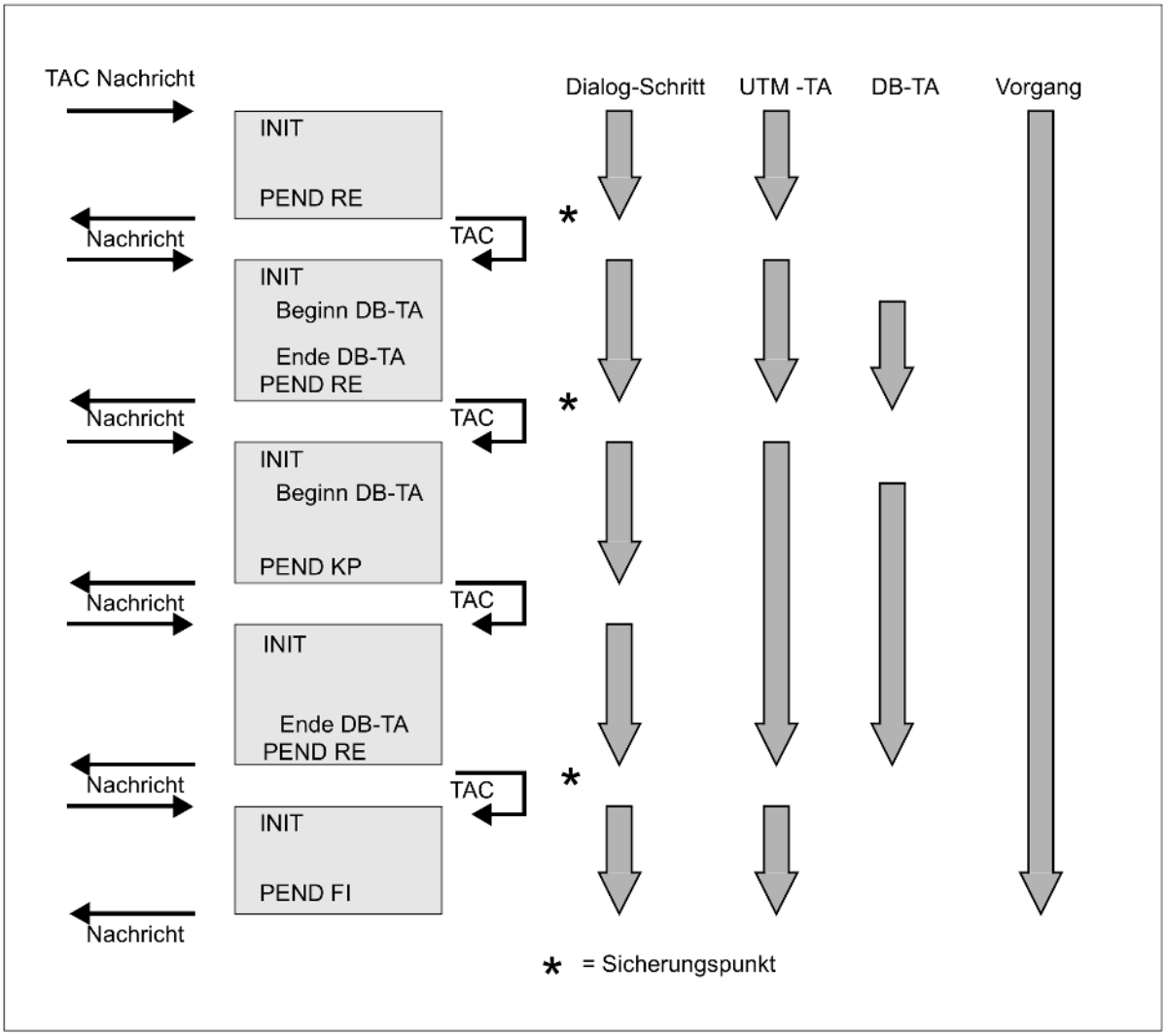


Bild: UTM-Transaktionen und DB-Transaktionen in einem Vorgang

> ¡ Der Aufruf PGWT CM verhält sich in Bezug auf Datenbank-Transaktionen wie PEND SP oder PEND RE.

Mehrschritt-Transaktionen

Eine Mehrschritt-Transaktion besteht aus mehreren UTM-Dialog-Schritten innerhalb einer Transaktion. Wenn man Datensätze im Dialog ändern möchte, ist das oft in einem Verarbeitungsschritt nicht möglich. Man braucht dazu einen ersten Verarbeitungsschritt zur Anzeige der Daten und einen Zweiten zum Eintragen der Änderungen. Die Sperre der Datensätze kann aufrechterhalten werden und über beide Verarbeitungsschritte zu einer Mehrschritt-Transaktion ausgedehnt werden.

Die Zeit zwischen den Verarbeitungsschritten wird von einem Timer überwacht, der bei der Generierung mit dem Generierungstool KDCDEF festgelegt wird. Nach Ablauf des Timers wird die Transaktion zurückgesetzt und die gesperrten Daten freigegeben.

Die Datenbanksysteme verwalten ihrerseits die Sperrdauer und lösen Sperren durch Rücksetzen von DB-Transaktionen auf. Der Vorgang wird beim nächsten DB-Aufruf darüber informiert.

Bei Mehrschritt-Transaktionen ist zu beachten, dass jedes Rücksetzen zur Wiederholung mehrerer Verarbeitungsschritte führen kann, je nachdem, wie weit der letzte Sicherungspunkt zurückliegt.

3.2 Programmierung von ESQL-Teilprogrammen

Ein UTM-ESQL-Teilprogramm ist wie ein normales UTM-Teilprogramm aufgebaut.

Für die Kommunikation mit der Datenbank stehen Ihnen alle Möglichkeiten der ESQL-Schnittstelle zur Verfügung.

Für die Transaktionssteuerung müssen Sie jedoch KDCS-Aufrufe verwenden. ESQL-Aufrufe wie BEGIN WORK, COMMIT WORK und ROLLBACK WORK sind bei der koordinierten Zusammenarbeit von openUTM mit Datenbanken nicht zugelassen. Weitere Einzelheiten finden Sie in den Handbüchern zu Ihrem Datenbanksystem.

> **i** Bei Kopplung über XA können geöffnete Cursor z.Zt. von Oracle nicht migriert werden, d.h. diese müssen explizit bei jedem Teilprogrammstart geöffnet und bei jedem Teilprogrammende geschlossen werden. Falls Sie die aktuelle Cursorposition im Folge-Teilprogramm benötigen, müssen Sie sich diese Position durch geeignete Programmierung merken (z.B. im KB oder im LSSB) und dann im Folge-Teilprogramm wieder aufsetzen.

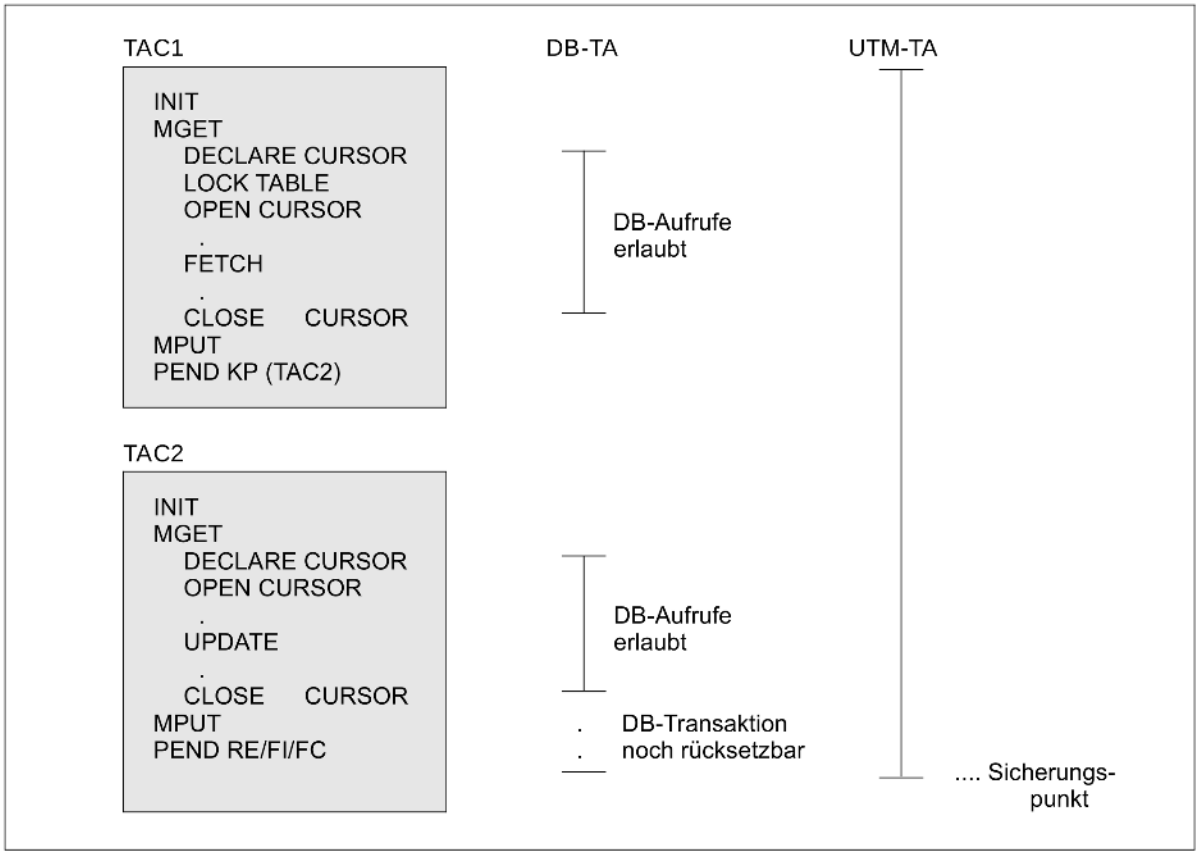


Bild: Beispiel für die Verwendung von CLOSE CURSOR

3.3 Fehlerbehandlung bei Datenbank-Kopplung

Treten bei der Zusammenarbeit mit Datenbanksystemen Fehler auf, so übernimmt openUTM die Fehlerbehandlung - der Programmierer muss keine speziellen Vorkehrungen treffen.

BS2000-Datenbanken ohne XA

Das Datenbanksystem informiert openUTM, ob es einen Aufruf erfolgreich durchführen konnte. Ist ein Fehler aufgetreten, so prüft openUTM den Fehlergrad und reagiert entsprechend: openUTM setzt die Transaktionen auf den letzten Sicherungspunkt zurück und gibt Meldungen aus, die Hinweise auf die Fehlerursachen enthalten.Tritt auf Grund der Zusammenarbeit mit der Datenbank ein schwerer Fehler auf, dann erzeugt openUTM die Meldung K071. Diese Meldung enthält Statusanzeigen der Datenbank und einen spezifischen Returncode, siehe openUTM-Handbuch „Meldungen, Test und Diagnose“.

Datenbankkopplung über XA

In diesem Fall informiert openUTM zusätzlich mit XA-Meldungen (K201 - K232) über den Status der Verbindung, außergewöhnliche Rollbacks, Commits und Fehler bei XA-Aufrufen (siehe auch openUTM-Handbuch „Meldungen, Test und Diagnose“).

Zur weiteren Diagnose steht ein spezieller Bereich des UTM-Dumps zur Verfügung, die UTM Diagarea. In diesen Bereich legt openUTM Trace-Informationen ab. Neben den KDCS-Aufrufen werden hier auch alle Aufrufe an das DB-System protokolliert. Diese Aufrufe finden Sie nach dem String DBCL (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

Verhalten nach Systemabsturz

Nach einem Absturz des Betriebssystem mit Abbruch der UTM-Anwendung sollten beim Neustart die Datenbanken vor der UTM-Anwendung gestartet werden. Wenn die UTM-Anwendung für die Variante UTM-S (Voreinstellung) generiert wurde, führt openUTM dann beim Start der Anwendung eine gemeinsame Recovery-Phase mit dem Datenbanksystem bzw. mit den Datenbanksystemen durch.

4 Bildschirmfunktionen

In diesem Kapitel finden Sie alle Informationen zu den Bildschirmfunktionen, die openUTM zur Verfügung stellt. Dazu gehören der Einsatz von Formaten, ebenso wie die Möglichkeit, im Zeilenmodus die Bildschirmausgabe zu beeinflussen. Außerdem beschreibt das Kapitel, wann openUTM für Terminals einen automatischen Bildschirmwiederanlauf durchführt, und wie Formatnamen beim Nachrichtenaustausch mit UPIC-Clients behandelt werden.

Programmierhinweis

Um das Programmieren zu erleichtern, stellt openUTM beim INIT-Aufruf im Feld KCRMF/kcrfn den Namen des Formats oder des Editprofils zur Verfügung, der beim Lesen der Nachricht mit MGET oder FGET angegeben werden muss.

Bei einem aus mehreren Teilformaten zusammengesetzten Format wird jedes Teilformat mit einem eigenen MGET-/FGET-Aufruf gelesen. Dabei liefert ein vorhergehender MGET-/FGET-Aufruf in KCRMF/kcrfn das Formatkennzeichen, das beim folgenden MGET-/FGET-Aufruf in KCMF/kcfn anzugeben ist.

> ¡ Auf Unix-, Linux- und Windows-Systemen wird kein Formatierungssystem unterstützt. Format-Namen können jedoch an UPIC-Clients und LU6.1-Partner übertragen werden.

4.1 Einsatz von Formaten in openUTM auf BS2000-Systemen

Ein Format - auch Maske genannt - ist nichts anderes als ein Formular, das an einem Terminal angezeigt oder auf einem Drucker ausgegeben wird. Wie jedes andere Formular besteht ein Format aus Feldern, die man ausfüllen kann (Eingabefelder), und aus Texten, die zum Formular gehören (Textfelder). Formate sind also Formulare, die in einem Rechner gespeichert sind und bei Bedarf an ein Terminal oder einem Drucker ausgegeben werden (siehe auch Abschnitt „Ausgaben auf Drucker“).

Ein Format enthält zusätzlich Angaben darüber, wie ein Feld auf dem Bildschirm dargestellt wird (z.B. blinkend), was man in ein Feld eintragen kann (z.B. nur numerische Werte) oder wo bei der Formatanzeige der Cursor stehen soll.

Zusammenarbeit mit dem Formatierungssystem

openUTM arbeitet mit dem Formatsteuersystem FHS (Format Handling System) über die Schnittstelle IUTMFORM zusammen.

Mit dem Formatgenerator IFG (Interactive Format Generator) können Sie FHS-Formate im geführten Dialog schnell und einfach erstellen. Dabei werden automatisch
Adressierungshilfen generiert, die Sie in den Teilprogrammen zur Strukturierung des Nachrichtenbereichs verwenden können. Die fertigen Formate werden in Bibliotheken abgelegt.

In den Teilprogrammen versorgen Sie bei Ein-/Ausgaben das Feld KCMF/kcfn mit dem Formatkennzeichen des gewünschten Formats. Die Nachrichten werden dann von openUTM in Zusammenarbeit mit dem Formatsteuersystem automatisch formatiert.

Das Formatkennzeichen setzt sich zusammen aus:

- dem Präfix (*, + oder #), das den Typ des Formats angibt (siehe nächsten Abschnitt). Außerdem ist das Zeichen "-" möglich.
- dem Formatnamen (höchstens 7 Zeichen lang).

Formattypen

openUTM unterscheidet *Formate, +Formate und #Formate:

*Formate	verwenden Sie dann, wenn Sie im Programm die Eigenschaften der Formatfelder (z.B. Anzeigeeigenschaften) **nicht** ändern wollen. Bei *Formaten werden bei Ein-/Ausgaben nur die Datenfelder übertragen.
+Formate	sind Formate, bei denen Sie im Programm die Eigenschaften einzelner Formatfelder abändern können. So können Sie z.B. eine falsche Eingabe blinkend an das Terminal zurückschicken. Hierzu wird jedem Datenfeld ein 2 Byte langes Attributfeld vorangestellt, in das im Programm die gewünschte Attributkombination eingetragen werden kann. Wird binär null eingetragen, so gilt die bei der Formaterstellung definierte Attributkombination.
	openUTM stellt Ihnen alle zulässigen Attributkombinationen in Sprach-spezifischen Datenstrukturen zur Verfügung - für COBOL im COPY-Element KCATC, für C/C++ in der Include-Datei *kcat.c*.
#Formate	sind Formate, bei denen Sie im Programm sowohl die Eigenschaften einzelner Formatfelder als auch globale Eigenschaften der Formatfelder abändern können. Bei der Ein-/Ausgabe sind Attributfelder und Datenfelder in getrennten Blöcke aufgeteilt. Näheres hierzu finden Sie im Handbuch des Formatierungssystems.

Außerdem gibt es -Formate. -Formate werden nicht vom Formatierungssystem formatiert sondern vom Event-Exit FORMAT (siehe "Event-Exit FORMAT (BS2000-Systeme)"). Falls das Formatkennzeichen mit dem Zeichen "-" beginnt verzweigt openUTM zu dieser vom Anwender selbsterstellten Formatierungsroutine.

Positionieren des Cursors

Bei formatierten Ausgaben mit +Formaten und *Formaten ist ein programmgesteuertes Positionieren des Cursors möglich (Funktion KDCSCUR).

Ob Sie für das Setzen des Cursors die Adresse des Attributfeldes oder des Datenfeldes angeben müssen, hängt vom FHS-Startparameter ATTR bzw. NOATTR ab:

- ATTR (nur zulässig bei +Formaten): Angabe des Attributfeldes
- NOATTR (bei *Formaten und +Formaten): Angabe des Datenfeldes

Der Cursor kann durch entsprechende Markierung des Attributfeldes an den Anfang des zugeordneten Datenfeldes gesetzt werden. Die Position des entsprechenden Bildschirmfeldes ist aus der Formatbeschreibung bekannt. Werden im Programmlauf mehrere Attributfelder einer Ausgabenachricht gleichzeitig für den Cursor gekennzeichnet, so wird nur der erste Eintrag berücksichtigt.

Durch den Aufruf der Funktion KDCSCUR() wird der Cursor auf ein bestimmtes Feld gesetzt. Als Argument der Funktion wird das Feld im Nachrichtenbereich angegeben, auf das der Cursor bei der nächsten Ausgabe gesetzt werden soll.

Setzen des Cursor in COBOL-Teilprogrammen:

```
CALL "KDCSCUR" USING FNAME.
```

Setzen des Cursor in C/C++-Teilprogrammen:

`KDCSCUR (feldname);` (Das Ergebnis der Funktion ist vom Typ void)

Im Zusammenhang mit KDCS-Attributfunktionen (+Formate und *Formate) gilt folgende Vereinbarung:
Durch den Aufruf eines Unterprogrammes "KDCSCUR", mit dem i-ten Attributfeld AF(i) als Parameter, kann der Cursor auf den Anfang des Nachrichtenfeldes F(i) positioniert werden. Das Unterprogramm enthält das Cursor-Kennzeichen als Konstante und setzt es additiv zu den bereits spezifizierten Attributen in das angegebene Attributfeld ab.

Beispiel:

```
CALL "KDCSCUR" USING AF1
```

Hier möchte der Teilprogrammlauf neben dem bereits definierten Feldattribut auch den Cursor an den Anfang des Nachrichtenfeldes F1 setzen.

Formatwechsel zwischen Ausgabe und Eingabe

Wenn openUTM eine Nachricht mit einem bestimmten Formatkennzeichen ausgegeben hat, dann muss bei der nachfolgenden Eingabe bei MGET in KCMF/kcfn der gleiche Formatname angegeben werden.

Ausnahme:
Falls nicht mit Teilformaten gearbeitet wird, toleriert openUTM beim MGET ein falsches Formatkennzeichen in KCMF/kcfn: Die Nachricht wird trotzdem gemäß dem letzten Bildschirmformat formatiert, der Returncode 05Z wird gesetzt und in KCRMF/kcrfn das Formatkennzeichen des zuletzt ausgegebenen Formats angezeigt.

Falls beim Einlesen von Teilformaten im Feld KCMF/kcfn ein falsches Formatkennzeichen angegeben ist, setzt openUTM den Returncode 03Z, gibt in KCRMF/kcrfn das richtige Formatkennzeichen zurück und setzt KCRLM auf 0. In den Nachrichtenbereich wird nichts eingetragen.

4.1.1 Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme)

openUTM bietet Ihnen die Möglichkeit, bei Verwendung von +Formaten und *Formaten zusammen mit einer Nachrichtenausgabe bestimmte Bildschirmfunktionen anzufordern. Dazu versorgen Sie das Feld KCDF (Device Function) des KDCS-Parameterbereichs mit einem der für die jeweilige Programmiersprache vordefinierten Werte. Diese befinden sich für C/C++ in der Include-Datei *kcdf.h* und für COBOL in dem COPY-Element KCDFC.
Bei #Formaten muss KCDF auf binär null gesetzt werden. Bei diesen Formaten erfolgt die Steuerung der Bildschirmfunktionen durch Globalattribute.

Die folgenden Funktionen können Sie bei +Formaten und *Formaten nutzen:

KCREPL (replace)

Der Bildschirm soll vor Ausgabe gelöscht und dann neu beschrieben werden.

KCERAS (erase)

Ist bei KCLM = 0 angegeben, werden alle variablen Felder gelöscht, das Format bleibt erhalten. Bei KCLM > 0 werden die modifizierten Daten im alten Format ausgegeben, übriggebliebene variable Felder werden gelöscht.

KCALARM (alarm)

Bei der Ausgabe ertönt ein akustisches Signal.

KCREPR (reproduce)

Ausgabe des Bildschirminhalts auf Drucker.

KCRESTRT (restart)

Bildschirmwiederanlauf nach PEND RS.

KCNODF (no device feature)

Keine Bildschirmfunktion, KCDF wird auf binär null gesetzt.

Wird mit Teilnachrichten gearbeitet, so gilt nur die Angabe bei der ersten Teilnachricht, für die folgenden Teilnachrichten muss das Feld KCDF den Wert binär null haben.

Sie können mehrere Bildschirmausgabefunktionen kombinieren, z.B. durch folgende Anweisung:

COBOL: KCDF = KCREPL + KCREPR + KCALARM
C/C++: kcdf = KCREPL | KCREPR | KCALARM

Die Funktion KCREPL sollten Sie allerdings aus Performancegründen sparsam verwenden.

Die Wirkung von KCERAS und KCREPL hängt von der Wahl der FHS-Startparameter ab, siehe „FHS-Benutzerhandbuch“.

Zum Thema Fehlerbehandlung bei Formatierungsfehlern siehe openUTM-Handbuch „Meldungen, Test und Diagnose“.

4.1.2 Vorgänge über Basisformate starten (BS2000-Systeme)

Falls gewünscht, können Formate bereits vor Beginn eines Vorgangs zur Verfügung gestellt werden, um die Eingabe der für den Vorgang erforderlichen Daten komfortabel zu gestalten.
Solche Formate werden **Basisformate** genannt. Für den Einsatz von Basisformaten gibt es verschiedene Möglichkeiten

- Startformate definieren
  Bei der Konfiguration können Sie jedem LTERM-Partner ein spezifisches Startformat zuordnen. In Anwendungen mit Event-EXIT SIGNON (siehe "Anmelde-Vorgang SIGNON") kann der Exit dieses Startformat lesen, wenn noch kein Benutzer an diesem LTERM angemeldet ist. In Anwendungen ohne Event-EXIT SIGNON und ohne Benutzerkennungen wird nach dem Verbindungsaufbau eines Terminals über diesen LTERM-Partner dieses Startformat ausgegeben.
  Bei der Konfiguration der UTM-Anwendung können Sie außerdem für jede Benutzerkennung ein Benutzer-spezifisches Startformat festlegen. In Anwendungen ohne Event-Exit SIGNON gibt openUTM dieses Format auf dem Bildschirm aus, nachdem sich ein Benutzer unter dieser Benutzerkennung bei openUTM angemeldet hat. In Anwendungen mit Event-EXIT SIGNON kann der Exit dieses Startformat lesen, wenn sich ein Benutzer unter dieser Benutzerkennung angemeldet hat. Als Startformate sind *Formate, +Formate und #Formate möglich. #Formate sind als Startformate jedoch nur bei Verwendung des Event-Service SIGNON zulässig.
- Formatausgabe zum Vorgangsende
  Sie können aus einem Teilprogramm am Ende eines Dialog-Vorgangs ein Format ausgeben, das dem Benutzer am Terminal dann zum Start des nächsten Vorgangs zur Verfügung steht.
- Format-Anforderung mit KDCFOR
  Der Benutzer am Terminal kann mit dem Benutzerkommando KDCFOR ein Basisformat anfordern. openUTM gibt dann das gewünschte Format aus. KDCFOR ist nicht erlaubt für #Formate.

Damit mit der Eingabe eines Formats ein Vorgang gestartet werden kann, muss zusammen mit der Nachricht auch der gewünschte Transaktionscode übergeben werden. Dafür gibt es folgende Möglichkeiten:

- Im Format ist als erstes Eingabefeld ein 8 Zeichen langes Feld vorgesehen, in das der Benutzer den gewünschten Transaktionscode einträgt.
  Beachten Sie, dass dieses Feld (bei +Formaten einschließlich Attributfeld) **nicht** in den Nachrichtenbereich übertragen wird (Ausnahmen: #Formate, im Event-Service BADTACS oder wenn mittels INPUT-Exit andere Festlegungen getroffen wurden). Enthält das Transaktionscode-Feld Leerzeichen, dann wird nur die Zeichenfolge bis zum ersten Leerzeichen als Transaktionscode interpretiert. Bei der Verwendung von Adressierungshilfen zur Strukturierung des Eingabebereichs müssen Sie die Abtrennung des Transaktionscodes berücksichtigen.
- Das Format sieht eine beliebige andere Stelle für die Angabe des Transaktionscodes vor. Die Eingabe wird mit dem Input-Exit ausgewertet und der Transaktionscode extrahiert.
- Der Transaktionscode ist per Generierung einer Funktionstaste zugeordnet, siehe KDCDEF-Anweisung SFUNC im openUTM-Handbuch „Anwendungen generieren“, und der Benutzer löst diese Funktionstaste aus.
- Die Funktionstaste muss eine F-Taste sein, da bei K-Tasten keine Nachricht mit übergeben wird (siehe auch KDCDEF-Anweisung SFUNC).
- Das Format enthält an beliebiger Stelle ein oder mehrere UTM-Steuerfelder. In ein Steuerfeld kann entweder der Benutzer den Transaktionscode eintragen oder es ist bereits mit einem Transaktionscode vorbelegt. Die Eigenschaft "UTM-Steuerfeld" wird bei der Formaterstellung vergeben.

Falls Sie Basisformate für den Vorgangsstart einsetzen, sollten Sie jeweils im ersten Teilprogramm das Feld KCRMF/kcrfn daraufhin auswerten, ob für den Vorgangsstart auch das vorgesehene Format verwendet wurde.

4.1.3 Einsatz von Teilformaten (BS2000-Systeme)

Eine Bildschirmausgabe kann aus mehreren Teilformaten aufgebaut werden. Teilformate belegen in der Regel nur einen Teil des Bildschirms. Für jedes Teilformat müssen Sie einen eigenen MPUT NT-Aufruf absetzen. Soll ein solcher Bildschirm wieder gelesen werden, dann ist für jedes Teilformat auch wieder ein eigener MGET-Aufruf nötig.

4.1.3.1 Ausgabeformatierung mit mehreren Teilformaten (BS2000-Systeme)

Wird ein Bildschirm aus mehreren Teilformaten neu aufgebaut, dann müssen Sie beim ersten MPUT NT im Feld KCDF den Wert KCREPL mit angeben. Bei allen folgenden MPUT-Aufrufen muss KCDF den Wert binär null haben, sonst bricht openUTM den Vorgang mit KCRCCC = 70Z und KCRCDC = K606 ab.

Formatmodus und Zeilenmodus können Sie innerhalb einer Dialog-Nachricht nicht mischen: Falls bei Teilnachrichten zwischen Zeilenmodus und Formatmodus gewechselt wird, dann wird der Vorgang mit KCRCCC=75Z abgebrochen.

Ein Wechsel zwischen *Formaten und +Formaten ist erlaubt.

Ändern eines Bildschirms

Sie können einen Bildschirm mit einem oder mehreren MPUT NT-Aufrufen ändern. Beim ersten MPUT-Aufruf dürfen Sie im Feld KCDF alle Werte angeben außer KCREPL, da mit KCREPL der gesamte Bildschirm gelöscht würde. Bei nachfolgenden MPUT NT-Aufrufen muss KCDF wie beim Neuaufbau auf binär null stehen.

Beim Formatwechsel löscht openUTM dann nur diejenigen alten Teilformate, die sich mit dem neuen überschneiden.

Mit der Bildschirmausgabefunktion KCERAS können Sie die variablen Felder eines Teilformats bei der Ausgabe löschen.

Ausgabe mehrerer Teilformate mit FPUT/DPUT NT

Einzelne Teilformate können mit FPUT/DPUT NT ausgegeben werden. Ein Bildschirm kann jedoch nicht mit mehreren FPUT/DPUT NT-Aufrufen aufgebaut werden, da openUTM bei Ausgaben auf Bildschirme jedes mit FPUT /DPUT NT gesendete Teilformat als eigene Nachricht überträgt. Der Wechsel zwischen Format- und Zeilenmodus ist erlaubt.

Vor Ausgabe einer formatierten Asynchron-Nachricht wird der Bildschirm automatisch gelöscht. Deshalb ist mit FPUT/DPUT NT der Update eines angezeigten Formats nicht möglich.

Eingaben aus asynchron ausgegebenen Formaten - außer Kommando-Eingaben - beantwortet openUTM mit einem automatischen Bildschirmwiederanlauf (siehe Abschnitt „Bildschirmwiederanlauf“).

Bei Ausgaben auf Drucker werden alle mit FPUT/DPUT NT gesendeten Teilnachrichten zusammen als eine Nachricht ausgegeben, auch dann, wenn zwischen Format- und Zeilenmodus gewechselt wird.

4.1.3.2 Eingabeformatierung mit mehreren Teilformaten (BS2000-Systeme)

Der Terminal-Benutzer kann Daten in die Teilformate eingeben, er muss aber nicht in jedes Teilformat etwas eintragen.

Da openUTM alle variablen Felder überträgt, enthält KCRMF/kcrfn nach dem INIT den Namen des ersten Teilformates, welches variable Felder enthält. Nach dem MGET steht in KCRMF/kcrfn der Name des nächsten Teilformats mit variablen Feldern. Nachdem das letzte Teilformat mit variablen Feldern gelesen wurde, ist KCMF = KCRMF (kcfn = kcrfn). Falls versucht wird, ein weiteres Teilformat zu lesen, obwohl die Nachricht bereits vollständig gelesen wurde, reagiert openUTM mit dem Returncode 10Z.

Beispiel

Im vorangehenden Teilprogrammlauf erfolgte eine MPUT-Ausgabe mit den 3 Teilformaten TEILF1, TEILF2, TEILF3; variable Felder enthalten nur TEILF1, TEILF2

KDCS-Aufruf:		**Rückgabe von openUTM:**
INIT		KCRMF = TEILF1
MGET	KCMF = TEILF1 KCLA = ...	KCRMF = TEILF2 KCRCCC = 000 KCRLM = ... Daten im Nachrichtenbereich
MGET	KCMF = TEILF3 KCLA = ...	KCRMF = TEILF2 KCRCCC = 03Z (weil KCMF einen anderen Wert enthält, als beim vorhergehenden MGET in KCRMF zurückgegeben wurde) KCRLM = 0
MGET	KCMF = TEILF2 KCLA = ...	KCRMF = TEILF2 KCRCCC = 000 KCRLM = ... Daten im Nachrichtenbereich
MGET	KCMF = ... KCLA = ...	KCRMF = TEILF2 KCRCCC = 10Z KCRLM = 0

Vorgangsstart durch Eingabe eines aus Teilformaten bestehenden Formats

Falls durch die Eingabe eines aus mehreren Teilformaten bestehenden Formats ein Vorgang gestartet werden soll, so kann für die Angabe des Transaktionscodes das erste variable Feld des ersten Teilformats vorgesehen werden (weitere Möglichkeiten siehe "Einsatz von Teilformaten (BS2000-Systeme)").
openUTM trennt dann automatisch das Feld mit dem Transaktionscode von der Nachricht ab: Bei *Formaten werden beim ersten Teilformat die ersten 8 Zeichen entfernt (Transaktionscode), bei +Formaten die ersten 10 Zeichen (Attributfeld und Transaktionscode).
Bei den weiteren Teilformaten wird nichts entfernt.

Aus einem aus Teilformaten bestehenden Format, das zum Ende eines Dialog-Vorgangs ausgegeben wurde, lässt sich auch ein Asynchron-Vorgang starten. Das Asynchron-Programm muss analog zum Dialog-Fall geschrieben werden: Der INIT-Aufruf liefert in KCRMF/kcrfn den Namen des ersten Teilformates mit variablen Feldern. Mit einem FGET-Aufruf kann das Programm die Eingabedaten aus diesem Teilformat holen. Bei einer leeren Eingabenachricht (Transaktionscode ohne Daten) liefert der FGET-Aufruf 10Z an das Teilprogramm zurück.

4.1.4 Nachrichtenfluss bei formatierten Nachrichten (BS2000-Systeme)

Der Weg der Nachrichten bei einem MPUT-Aufruf an ein Terminal ist in folgendem Bild dargestellt:

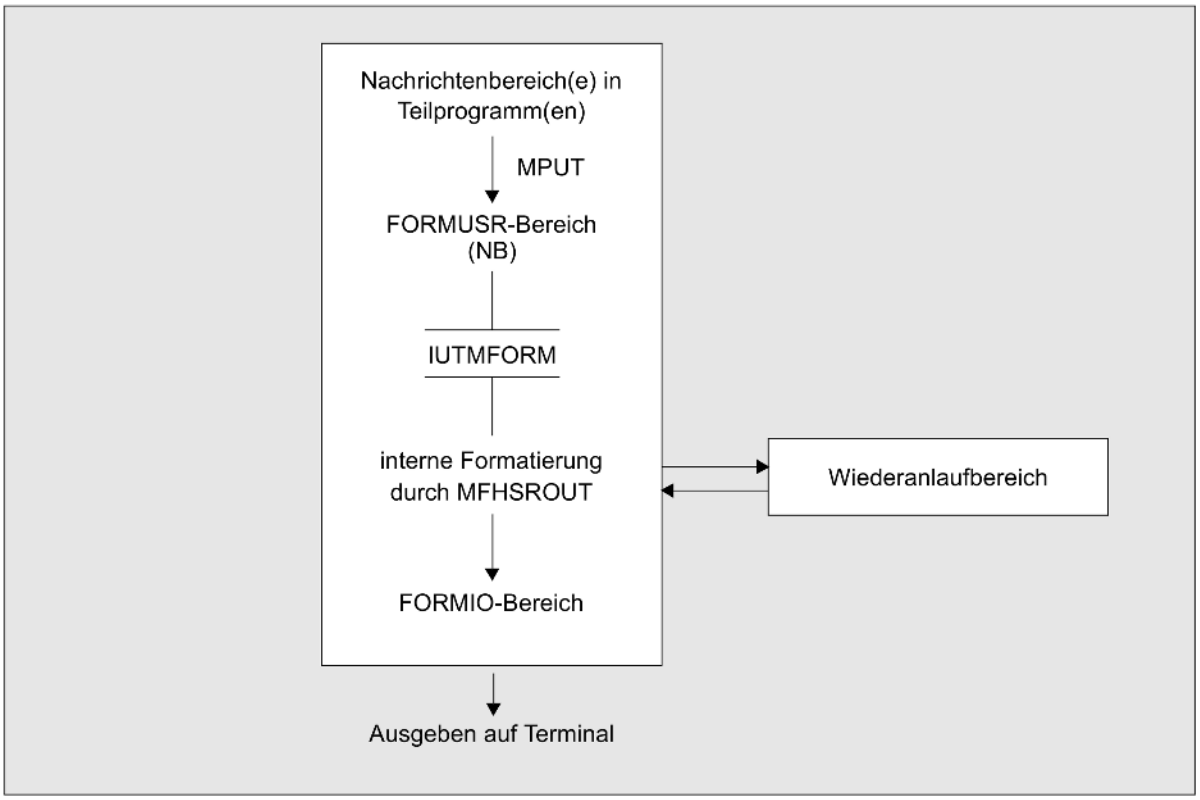


Bild: Nachrichtenfluss bei MPUT an Terminal

Die Nachrichten aus den Teilprogrammen kommen zur Formatierung einer Ausgabenachricht über UTM-Zwischenspeicher in den FORMUSR-Bereich. Die Größe dieses Bereichs, der die gesamte logische Nachricht aufnehmen können muss, wird im Operand NB der MAX-Anweisung vom Anwender festgelegt (siehe openUTM-Handbuch „Anwendungen generieren“). Diese Nachricht wird vom FHS-Formatierungsprogramm MFHSROUT formatiert, die formatierte Nachricht im FORMIO-Bereich abgelegt. Diese Nachricht wird an das Terminal gesendet.

Wird mit Teilformaten gearbeitet, so muss der Bereich FORMUSER immer die gesamte Nachricht aufnehmen können.

Eingabeformatierung

Bei der Eingabeformatierung läuft der Nachrichtenstrom in umgekehrter Richtung, wobei die gleichen Bereiche verwendet werden.

4.1.5 Ausgaben auf Drucker im Formatmodus (BS2000-Systeme)

Auch auf Drucker können Formate ausgegeben werden. Informationen zu den unterstützten Druckerfunktionen finden Sie in den Handbüchern Ihres
Formatierungssystems.

Im Feld KCMF/kcfn des KDCS-Parameterbereichs tragen Sie beim FPUT/DPUT-Aufruf ein:

*formatname | +formatname | #formatname

Dabei bedeutet:

*	Formatierung ohne Möglichkeit einer Attributänderung
+	Formatierung mit der Möglichkeit, die Attribute einzelner Formatfelder zu ändern
#	Formatierung mit der Möglichkeit, sowohl die Eigenschaften einzelner Formatfelder als auch globale Eigenschaften der Formatfelder abzuändern

Das Formatierungssystem muss die Druckertypen unterstützen wie sie im VTSU-B bekannt bzw. bei RSO (bei PTYPE=*RSO) generiert sind.

Falls Sie für den Ausdruck den Event-Exit FORMAT einsetzen, müssen Sie folgende Punkte beachten:

- Der Restartbereich muss nicht versorgt werden.
- Im Nachrichtenkopf ist die Abdruckquittung anzufordern.
- Im Nachrichtenkopf sind die von openUTM übergebenen Quittungs-Bytes einzutragen.
- Angaben für die Formatierung, wie Formatname, Gerätetyp usw. entnehmen Sie dem KB-Kopf und den Parameterbereichen des Event-Service FORMAT.

Den Aufbau des Nachrichtenkopfes finden Sie in den Handbüchern zu Ihrem Drucker.

Formularvorschub

Der Formularvorschub hängt bei allen FPUT/DPUT-Aufrufen, auch bei Teilnachrichten mit FPUT/DPUT NT, vom Inhalt des Feldes KCDF im KDCS-Parameterbereich ab (Ausnahme: Verwendung von #Formaten):

KCREPL	Formularvorschub vor dem Drucken, d.h. das Format wird ab der am Drucker eingestellten Formulargrundstellung gedruckt.
sonst	Kein Formularvorschub, d.h. das Format wird ab der nächsten Zeile ausgedruckt (auch beim ersten FPUT/DPUT).

Damit können Sie Formate aneinanderhängen, um umfangreiche Formulare

auszudrucken, unabhängig von Längenbeschränkungen. Die physische Teilnachricht muss aber kürzer sein als die:

- BCAM-Letterlänge (siehe TRANSDATA-Handbuch „Generierung eines Datenkommunikationssystems“) und
- die Geräte-spezifische Länge, die von VTSU-B beim Verbindungsaufbau geliefert wird.

4.2 Beeinflussen der Ausgabe im Zeilenmodus (BS2000- Systeme)

openUTM arbeitet immer dann im Zeilenmodus, wenn der in KCMF/kcfn angegebene Name mit einem Leerzeichen beginnt. Auch im Zeilenmodus bietet Ihnen openUTM einige Möglichkeiten, die Bildschirmausgabe zu beeinflussen:

- Strukturieren der Ausgabe mit logischen Steuerzeichen
- Zusätzlich zu den auch im Formatmodus nutzbaren Bildschirmausgabefunktionen KCALARM, KCREPR, KCRESTRT (siehe "Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme)") sind im Zeilenmodus auch die Bildschirmausgabefunktionen KCCARD und KCEXTEND (erweiterter Zeilenmodus) nutzbar.
- Verwendung von Editprofilen

Wie Sie Nachrichten an Drucker aufbereiten, ist im Abschnitt „Ausgaben auf Drucker“ beschrieben.

Strukturieren mit logischen Steuerzeichen

Im Zeilenmodus können Sie Ihre Ausgabe mit logischen Steuerzeichen strukturieren. Die Terminalunterstützung VTSU (Virtual Terminal Support) wandelt diese Steuerzeichen dann in die physischen Steuerzeichen um, die das entsprechende Gerät benötigt. Sie können dazu alle logischen Steuerzeichen verwenden, die auch die Zugriffsmethode TIAM zulässt (siehe „TIAM Benutzerhandbuch“). Für einige Programmiersprachen gibt es Datenstrukturen, die in das Programm kopiert werden können (siehe „TIAM-Benutzerhandbuch“).

Bildschirmfunktion KCCARD

Die Bildschirmfunktion KCCARD ermöglicht die Dateneingabe durch Magnetstreifenkarte: Bei Ausgabe einer Dialog-Nachricht mit der Bildschirmfunktion KCCARD wird die Tastatur gesperrt und der Terminalbenutzer aufgefordert, eine Magnetstreifenkarte in den Ausweisleser einzulegen.

Bildschirmfunktion KCEXTEND

Mit dieser Bildschirmfunktion werden die Felder bei der Ausgabe mit einer bestimmten Voreinstellung versehen, die dem Makro WRTRD, ... EXTEND=YES entspricht. Alle ausgegebenen Felder sind standardmäßig halbhell und geschützt (siehe „TIAM Benutzerhandbuch“).

Editprofile

Als Editprofil bezeichnet man einen Satz von Eigenschaften für die Ausgabe im Zeilenmodus. Mit Editprofilen kann man z.B. festlegen, dass eingegebene Zeichen am Terminal nicht dargestellt werden (Passworteingabe), oder die Umwandlung von Klein- in Großbuchstaben anfordern. Neben der Bildschirmfunktion KCCARD bieten Editprofile eine weitere Möglichkeit, Eingaben vom Ausweisleser anzufordern. Auch andere Bildschirmfunktionen, wie z.B. KCREPR, lassen sich mit Editprofilen realisieren.

Der Name und die Eigenschaften eines Editprofils werden beim Generieren mit der KDCDEF-Anweisung EDIT vergeben (siehe openUTM-Handbuch „Anwendungen generieren“). Der Name eines Editprofils ist bis zu sieben Zeichen lang. Ein Editprofil wird im Teilprogramm beim MPUT/FPUT/DPUT-Aufruf angesprochen, indem Sie in das Feld KCMF/kcfn im ersten Byte ein Leerzeichen und in die restlichen Bytes den Namen des Editprofils eintragen. Editprofile werden wie Formatnamen behandelt, d.h. der Name des Editprofils der letzten Ausgabenachricht wird nach dem INIT im Feld KCRMF/kcrfn zurückgegeben (wie beim Formatkennzeichen). Für MGET/FGET-Aufrufe wird er in das Feld KCMF/kcfn eingetragen.

Wenn Sie Editprofile verwenden, dann müssen Sie Folgendes beachten:

- Bei MPUT-, FPUT- oder DPUT-Aufrufen mit Editprofilen darf keine Bildschirmfunktion angegeben werden (KCDF muss binär null enthalten), sonst reagiert openUTM mit 70Z bei MPUT oder mit 40Z bei FPUT/DPUT.

- Sobald für eine Anwendung Editprofile generiert sind, müssen Sie Nachrichten im reinen Zeilenmodus (ohne Editprofil) mit acht Leerzeichen in KCMF/kcfn kennzeichnen; es reicht nicht, nur das erste Byte mit Leerzeichen zu versorgen, da openUTM dies wie ein falsches Formatkennzeichen behandelt und den entsprechenden Fehlercode in KCRCCC setzt.
- Sind mehrere Teilnachrichten mit MPUT NT, FPUT NT oder DPUT NT an ein Terminal gerichtet, darf der Name des Editprofils nicht wechseln, sonst quittiert openUTM dies mit 75Z bei MPUT oder mit 45Z bei FPUT/DPUT. Bei Teilnachrichten an Drucker darf gewechselt werden.
- Wurde der Bildschirm durch Ausgabe einer Asynchron-Nachricht überschrieben, dann führt openUTM bei Editprofilen bei der nächsten Eingabe einen automatischen KDCDISP durch.
- Bei Nachrichten an UPIC-Clients werden die Namen der Editprofile wie Formatkennzeichen behandelt, d.h. sie werden mit übertragen (erscheinen also in im UTM-Teilprogramm in KCRMF/kcrfn), bleiben aber wirkungslos.
- Bei Nachrichten über verteilte Verarbeitung werden die Namen der Editprofile wie Formatkennzeichen behandelt:
  - Bei verteilter Verarbeitung über LU6.1 werden die Namen der Editprofile zwar mit übertragen (d.h. sie erscheinen in KCRMF/kcrfn), bleiben aber wirkungslos.
  - Bei verteilter Verarbeitung über OSI TP darf in KCMF/kcfn kein Editprofil angegeben werden, da openUTM das Feld des Formatkennzeichens zur Übergabe der abstrakten Syntax verwendet.

4.3 Ausgaben auf Drucker im Zeilenmodus

Wenn Sie im Zeilenmodus ausgeben wollen, muss das erste Byte im Feld KCMF/kcfn des KDCS-Parameterbereichs ein Leerzeichen sein.

Im Zeilenmodus darf die Nachricht alle logischen Steuerzeichen enthalten, z.B. für Formularvorschub oder Zeilenvorschub. Diese Steuerzeichen können Sie sich im Teilprogramm selbst definieren und damit den Text für die Druckausgabe selbst strukturieren.

Auf BS2000-Systemen haben Sie zusätzlich die Möglichkeit, für die Darstellung/Aufbereitung der Nachrichten Editprofile zu verwenden (siehe auch Abschnitt Editprofile in Kapitel „Beeinflussen der Ausgabe im Zeilenmodus (BS2000- Systeme)"). Den Namen des Editprofils tragen Sie in KCMF/kcfn ab dem zweiten Byte ein.

> **i** Eine Nachricht, abgeschlossen mit FPUT/DPUT NE, wird immer als eigenständige Nachricht ausgedruckt. Im Fall der Einzelblattzuführung muss der Benutzer bei Ausgaben im Zeilenmodus per Programm selbst für den Blattwechsel (Formularvorschub) sorgen, da das Gerät das Blattende nicht erkennen kann.

4.4 Bildschirmwiederanlauf

openUTM führt für Terminals in folgenden Fällen einen automatischen Bildschirmwiederanlauf durch:

- Wenn am letzten Sicherungspunkt eines Vorgangs eine Bildschirmausgabe erfolgte, und der Benutzer meldet sich nach Unterbrechung des Vorgangs erneut an der Anwendung an, z.B nach:
  - normaler Beendigung der Anwendung, ohne dass der Vorgang beendet wurde
  - abnormaler Beendigung der Anwendung
  - KDCOFF innerhalb eines Vorgangs
  - Verbindungsverlust

  Dieser Bildschirmwiederanlauf wird beim erneuten Anmelden nur durchgeführt, wenn die Benutzerkennung bzw. in Anwendungen ohne Benutzerkennungen der LTERM-Partner mit Wiederanlauf generiert ist.

  Setzen Sie dazu bei der Generierung in den entsprechenden USER- bzw. LTERM-Anweisungen den Operanden RESTART=YES (siehe openUTM-Handbuch „Anwendungen generieren“).

  Nach einem Anwendungsende ist in **stand-alone UTM-Anwendungen** ein Bildschirmwiederanlauf nur in UTM-S-Anwendungen möglich.
- Während eines Vorgangs wurde eine Asynchron-Nachricht auf dem Bildschirm ausgegeben und der Benutzer unterlässt es, zur Fortsetzung des Dialogs ein KDCDISP-Kommando abzusetzen, d.h. er versucht die Eingabe einer Nachricht in den asynchron ausgegebenen Bildschirminhalt (möglich sind nur Kommando-Eingaben).
- Innerhalb eines Vorgangs, nachdem eine Folgetransaktion durch MPUT RM, KCDF = KCRESTRT und PEND RS zurückgesetzt wurde, und am letzten
  Sicherungspunkt des Vorgangs eine Bildschirmausgabe erfolgte.
- Auf BS2000-Systemen bei bestimmten Bedienungsfehlern (z.B. Aus-, Einschalten des Terminals, AM-Taste).

Der Bildschirmwiederanlauf ist nur deshalb möglich, weil openUTM die Informationen über den letzten Bildschirmaufbau speichert, und zwar zunächst in einem Puffer im Prozess-spezifischen Systemspeicherbereich. Bei Transaktionsende schreibt openUTM diese Informationen in den Wiederanlaufbereich im Pagepool der KDCFILE, siehe openUTM-Handbuch „Anwendungen generieren“.

Bei der Ausgabeformatierung auf BS2000-Systemen schreibt openUTM nur dann den gesamten Bildschirminhalt in den Wiederanlaufbereich, wenn der Bildschirm komplett neu aufgebaut wird (KCDF= REPLACE, Formatwechsel). Bei einem Update des Bildschirms werden im Wiederanlaufbereich nur die Felder aktualisiert, die auch am Terminal überschrieben werden. Analog werden bei der Eingabeformatierung nur die Felder im Wiederanlaufbereich aktualisiert, die vom Terminal eingehen.

Wenn Sie auf einem BS2000-System ein Formatierungssystem nutzen, werden Aufbau, Ändern und Sichern des Wiederanlaufbereichs von openUTM in Zusammenarbeit mit dem Formatierungssystem automatisch durchgeführt. Wenn Sie dagegen mit -Formaten und selbst programmierter Formatierung arbeiten, übernimmt openUTM zwar das Sichern des Wiederanlaufbereichs, für Aufbau und Ändern des Bereichs ist jedoch die selbst programmierte Formatierungsroutine zuständig (siehe Event-Exit FORMAT, "Event-Exit FORMAT (BS2000-Systeme)").

4.5 Formatnamen beim Nachrichtenaustausch mit UPIC-Clients

Teilprogramme, die für den formatierten Dialog mit Terminals geschrieben sind, können Sie ohne Änderung auch für den Dialog mit UPIC-Clients einsetzen. In diesem Fall wird aber kein Formatierungssystem aufgerufen.
Zwischen openUTM und dem UPIC-Client werden Netto-(Teil-)Nachrichten und Formatnamen ausgetauscht. Beim Lesen der Nachricht verhält sich openUTM wie beim Lesen von Teilformaten, d.h. wird beim MGET ein falsches Formatkennzeichen angegeben, setzt openUTM den Returncode 03Z und gibt in KCRMF/kcrfn das richtige Formatkennzeichen zurück und setzt KCRLM auf 0. In den Nachrichtenbereich wird nichts eingetragen.
Die Bildschirmoberfläche kann der UPIC-Client dann z.B. anhand des Formatnamen aufbauen.

Auch ein programmgesteuertes Positionieren des Cursors mit "KDCSCUR" ist bei der Ausgabe einer Nachricht an den UPIC-Client möglich.

Ein UPIC-Client kann einer UTM-Anwendung den Wert einer Funktionstaste übergeben, so dass beim MGET im Teilprogramm der für diese Funktionstaste generierte Returncode ausgelöst wird (siehe Generierung SFUNC-Anweisung, Parameter RET).

Näheres siehe Handbuch „openUTM-Client für das Trägersystem UPIC“.

5 Programmaufbau bei verteilter Verarbeitung

Unter dem Begriff "verteilte Verarbeitung" werden alle Formen der Verarbeitung zusammengefasst, bei denen gleichberechtigte Server-Anwendungen zusammenarbeiten (Server-Server-Kommunikation). openUTM ermöglicht verteilte Verarbeitung über die Protokolle LU6.1, LU6.2 und OSI TP. Partner einer UTM-Anwendung können bei verteilter Verarbeitung andere UTM-Anwendungen, OpenCPIC-Anwendungen oder auch Anwendungen sein, die auf Transaktionssystemen anderer Hersteller basieren.

Die Programmschnittstellen für die Kommunikation über LU6.2 sind identisch zu denen für die Kommunikation über OSI TP. Zur Kommunikation über das Protokoll LU6.2 ist das Zusatzprodukt openUTM-LU6.2 erforderlich.

Mit dem Protokoll OSI TP kann wahlweise mit globaler Transaktionssicherung gearbeitet werden, oder auch ohne globale Transaktionen.

Bei Einsatz des Protokolls LU6.1 arbeitet openUTM grundsätzlich mit globaler Transaktionssicherung, d.h. die Transaktionen aller beteiligter Partner werden synchronisiert.

Client-Server-Kommunikation wird nicht als verteilte Verarbeitung bezeichnet, obwohl auch bei Client-Server-Kommunikation der Client Verarbeitungsaufgaben übernimmt, z.B. Plausibilitätsprüfungen durchführt, aber: die Client und die Server-Rolle sind jeweils fest verteilt. Der Client stellt keine Services zur Verfügung, die von anderen Anwendungen genutzt werden könnten.

Die Kommunikation mit UTM-Client-Anwendung mit Trägersystem OpenCPIC erfolgt über das Protokoll OSI TP; damit gelten für die Kommunikation mit diesen Partnern alle Aussagen und Regeln, die in diesem Handbuch für die Kommunikation mit OSI TP-Partnern getroffen werden, und zwar auch dann, wenn mit einer OpenCPIC-Anwendung ein Client einer UTM-Anwendung realisiert wird.

5.1 Adressierung ferner Vorgänge

Bevor in einem Vorgang eine Nachricht an einen fernen Vorgang geschickt werden kann, muss dieser ferne Vorgang adressiert werden. Hierzu dient der KDCS-Aufruf **APRO** (**A**ddress **PRO**gram).

Bei verteilter Verarbeitung können Sie sowohl die Dienste ferner Dialog-Vorgänge nutzen als auch - durch die Message Queuing-Funktionalität von openUTM - die Dienste ferner Asynchron-Vorgänge. Ferne Dialog-Vorgänge werden mit **APRO DM** (**D**ialog **M**essage) adressiert, ferne Asynchron-Vorgänge mit **APRO AM** (**A**synchronous **M**essage).

Wird in einer Transaktion mit APRO AM/DM ein ferner Vorgang adressiert, dann muss dem fernen Vorgang in der gleichen Transaktion auch eine Nachricht geschickt werden.

Über OSI TP ist es unter bestimmten Bedingungen auch möglich, mit APRO AM einen Dialog-Vorgang zu adressieren, also einen Asynchron-Auftrag an einen Dialog-Vorgang zu richten (siehe "Auftragnehmer-Seite").

Für die Adressierung eines fernen Vorgangs sind jeweils folgende Parameter des APRO-Aufrufs relevant:

- Im Feld **KCRN** (**R**eferenced **N**ame) geben Sie den LTAC-Namen des fernen Service an. Der LTAC-Name (Local TransAction Code) ist der Name, unter dem der ferne Service in der Konfiguration der lokalen Anwendung bekannt ist. Der LTAC-Name wird bei der Generierung vergeben.

  Falls dieser LTAC-Name bereits in der Konfiguration fest mit einer bestimmten Partner-Anwendung verknüpft ist, dann genügt beim APRO-Aufruf bereits die Angabe dieses Namens, um den gewünschten fernen Service eindeutig zu bestimmen. In diesem Fall spricht man von einstufiger Adressierung. Einstufige Adressierung ist immer dann sinnvoll, wenn es bezüglich der Partner-Anwendung keine Alternativen gibt, z.B. weil der angeforderte Service nur von einer einzigen Partner-Anwendung zur Verfügung gestellt wird.

Bei zweistufiger Adressierung wird die Partner-Anwendung beim APRO-Aufruf explizit festgelegt:

- Das Feld **KCPA** (**P**artner **A**pplication) dient bei zweistufiger Adressierung zur Angabe des (OSI-)LPAP-Namens der Partner-Anwendung. Der (OSI-)LPAP-Name (Logical Partner APplication) ist der Name, unter dem die Partner-Anwendung in der Konfiguration der lokalen Anwendung bekannt ist. Der (OSI-)LPAP-Name wird bei der Generierung vergeben. Im Feld KCPA kann auch der Name eines MASTER-LU61-LPAPs oder MASTER-OSI-LPAPs angegeben werden.

  Zweistufige Adressierung ist immer dann sinnvoll, wenn der angeforderte Service von mehreren Partner-Anwendungen zur Verfügung gestellt wird und die Auswahl der Anwendung von der konkreten Situation abhängt.

  Wird in KCPA der (OSI-)LPAP-Name einer Partner-Anwendung angegeben, obwohl der in KCRN spezifizierte LTAC-Name bereits in der Konfiguration einer Partner-Anwendung zugeordnet ist, so hat die Angabe in KCPA Vorrang vor der Konfigurationszuordnung.

- Im Feld **KCPI** (Partner Identification) ordnen Sie dem fernen Vorgang für die Dauer der Zusammenarbeit eine Identifikation zu. Diese Identifikation, auch Vorgangs-Id genannt, ist frei wählbar, das erste Zeichen muss jedoch ">" sein. Der Gültigkeitsbereich der Vorgangs-Id ist der lokale Vorgang; d.h. wenn zwei Vorgänge zu einer Zeit die gleiche Vorgangs-Id verwenden, dann adressieren sie damit unterschiedliche ferne Vorgänge. Bei allen Kommunikationsaufrufen, die den adressierten Vorgang betreffen, geben Sie diese Identifikation jeweils im Feld KCRN an. Sobald die Zusammenarbeit mit dem fernen Vorgang beendet ist, kann die Vorgangs-Id in weiteren APRO-Aufrufen zur erneuten Identifizierung des gleichen Service oder auch zur Identifizierung anderer Services verwendet werden.

  Die Aufrufe, bei denen Sie im Feld KCRN die Vorgangs-Id angeben müssen, sind im Einzelnen:

  - MPUT-Aufrufe zum Senden von Dialog-Nachrichten an den fernen Vorgang
  - FPUT- und DPUT-Aufrufe zum Senden von Asynchron-Nachrichten an den fernen Vorgang
  - MGET-Aufrufe zum Lesen von Dialog-Nachrichten oder Statusinformationen vom fernen Vorgang
  - MCOM BC-Aufrufe für Beginn eines Auftrags-Komplexes, dessen Basisauftrag an den fernen Vorgang gerichtet ist
  - CTRL-Aufrufe zum Steuern von OSI TP-Dialogen

5.2 Verteilte Dialoge

Auch bei verteilter Verarbeitung gilt die Regel des strengen Dialogs, d.h., bei der Kommunikation zwischen zwei Partnern muss, nachdem eine Nachricht an den Partner gesendet wurde, eine Antwort vom Partner eingehen, bevor erneut eine Nachricht an diesen Partner gesendet werden kann.

Der Zeitraum zwischen dem Empfangen einer Dialog-Nachricht und dem Senden der nächsten Dialog-Nachricht wird als **Verarbeitungsschritt** bezeichnet. Diese nächste Nachricht kann entweder eine Antwort sein - dann entspricht der Verarbeitungsschritt einem **Dialog-Schritt** - oder auch eine Nachricht an einen anderen Partner.

Ein Auftraggeber kann gleichzeitig mit mehreren Auftragnehmern zusammenarbeiten. Der nächste Verarbeitungsschritt im Auftraggeber beginnt erst, wenn alle Antworten von den Auftragnehmern eingetroffen sind. Für jeden Verarbeitungsschritt wird i.a. ein eigener Teilprogrammlauf gestartet (Ausnahme: PGWT).

Der Auftraggeber-Vorgang kann sowohl ein Dialog- als auch ein Asynchron-Vorgang sein.

Austausch von Dialog-Nachrichten

Dialog-Nachrichten zwischen Partner-Vorgängen werden - wie Dialog-Nachrichten ohne verteilte Verarbeitung - mit MPUT gesendet und mit MGET empfangen.

Beim Nachrichtenaustausch mit dem Auftraggeber verhält sich ein Auftragnehmer-Vorgang so, als würde er mit einem Terminal kommunizieren: In das Feld KCRN sind beim MGET und MPUT Leerzeichen einzutragen.

Ein Auftraggeber-Vorgang muss beim Nachrichtenaustausch mit einem Auftragnehmer im Feld KCRN die Vorgangs-Id des Auftragnehmers angeben, die beim APRO-Aufruf gewählt wurde (siehe ''Adressierung ferner Vorgänge'').

5.2.1 Steuern der Kommunikation im Programm

Bei verteilter Verarbeitung wird eine Aufgabe von mehreren Teilprogrammen in verschiedenen Partner-Anwendungen bearbeitet. Ein Teilprogramm kann von Clients (Terminals oder Client-Programmen), von Programmen der eigenen Anwendung und von fremden Anwendungen aus angesprochen werden. Es muss sich daher abhängig vom Partner entscheiden, welche Aufgabe es bearbeitet, an wen es die Nachricht schickt und ob es eine verteilte Verarbeitung beenden muss oder nicht.

Deshalb stellt openUTM den Teilprogrammen eine Reihe von Informationen über den Kommunikationspartner und den aktuellen Stand der Kommunikation zur Verfügung, die ausgewertet und für die Steuerung der Kommunikation verwendet werden können. Diese Informationen sind in den Abschnitten „Programmierhilfen" für LU6.1 und „Programmierhilfen" für OSI TP beschrieben.

5.2.2 Fehlerbehandlung im Teilprogramm

Es gibt zwei Situationen, in denen ein Teilprogramm auf Fehler reagieren kann:

- Bei weniger schweren Fehlern:
  Das Teilprogramm behält die Steuerung und kann mit programmiertem Rücksetzen reagieren (siehe unten).
- Nach einem Vorgangs-Wiederanlauf:
  Das Teilprogramm wird neu gestartet, erhält eine Statusinformation und evtl. eine Rücksetznachricht und kann auf die Fehlersituation reagieren.

5.2.2.1 Programmiertes Rücksetzen

Bei weniger schweren Fehlern behält das Teilprogramm die Steuerung und kann mit PEND RS oder PEND ER/FR die verteilte Transaktion oder mit RSET die lokale Transaktion zurücksetzen. Bei der Kommunikation über OSI TP kann eine Transaktion auch mit PGWT RB zurückgesetzt werden, siehe "Besonderheiten bei Rollback und Wiederanlauf".

Die folgenden Aussagen gelten für verteilte Verarbeitung mit globaler Transaktionssicherung, d.h. für LU6.1 und für OSI TP mit Functional Unit Commit. Wie diese Aufrufe bei OSI TP ohne Functional Unit Commit wirken, erfahren Sie auf "Besonderheiten bei Rollback und Wiederanlauf".

PEND RS

PEND RS kann in einem Auftraggeber- oder in einem Auftragnehmer-Vorgang aufgerufen werden.

Ein PEND RS wirkt in einer verteilten Transaktion wie folgt:

Mit PEND RS werden alle an einer verteilten Transaktion beteiligten Vorgänge auf den letzten Sicherungspunkt zurückgesetzt. Im Gegensatz zum PEND ER/FR-Aufruf bleibt beim PEND RS ein Vorgang offen, wenn er schon einen Sicherungspunkt erreicht hat. Vorgänge, die noch keinen Sicherungspunkt erreicht haben, werden beendet.

Es sind folgende Situationen möglich:

- PEND RS in der ersten Transaktion des obersten Auftraggeber-Vorgangs

  Alle beteiligten Vorgänge werden beendet, ohne dass ein Vorgangs-Wiederanlauf stattfindet. Gekettete Vorgänge werden nach dem Rücksetzen erneut gestartet.
- PEND RS in der ersten Transaktion eines Auftragnehmer-Vorgangs

  Der Auftragnehmer-Vorgang wird beendet. Wenn der oberste Auftraggeber-Vorgang schon einen Sicherungspunkt erreicht hat, dann führt openUTM einen Vorgangs-Wiederanlauf durch (mit der Meldung K034 an das Terminal). Der Auftraggeber-Vorgang erhält eine Statusinformation (Vorgangs-Status "R" / "Z" und Transaktionsstatus "R"), die mit MGET NT gelesen werden muss.

- PEND RS in einer Folge-Transaktion des Auftraggeber- oder Auftragnehmer-Vorgangs

  In diesem Fall muss vor dem PEND RS mit MPUT RM eine Rücksetznachricht gesendet werden, sonst bricht openUTM den Vorgang mit KCRCCC = 83Z ab. Die Rücksetznachricht geht an das Folge-Teilprogramm, das am letzten Sicherungspunkt des (lokalen) Vorgangs angegeben wurde. Nach dem Rücksetzen führt openUTM einen Vorgangs-Wiederanlauf durch. Die Art und Weise des Vorgangs-Wiederanlaufs hängt davon ab, an wen die Ausgabenachricht am letzten Sicherungspunkt ging und wer PEND RS aufgerufen hat:

  - Bei einer **Ausgabenachricht an den Client** beginnt der Vorgangs-Wiederanlauf im Auftraggeber-Vorgang.
    - Wurde PEND RS im obersten Auftraggeber-Vorgang abgesetzt, dann wird ein Bildschirmwiederanlauf durchgeführt (wie ohne Einsatz der verteilten Verarbeitung).
    - Wurde PEND RS in einem Auftragnehmer-Vorgang abgesetzt, dann wird ein Bildschirmwiederanlauf durchgeführt und die Meldung K034 ausgegeben. Die anschließende Eingabe vom Client wird vom Folge-Teilprogramm des letzten Sicherungspunkts gelesen. Sendet ein Folge-Teilprogramm erneut eine Nachricht an den Auftragnehmer (der PEND RS durchgeführt hat), dann liest dieser Auftragnehmer zuerst die Rücksetznachricht und dann die an ihn gesendete Nachricht. Wenn Auftragnehmer-Vorgänge durch das Rücksetzen beendet wurden (d.h. der Auftragnehmer war seinerseits wieder Auftraggeber), dann müssen zusätzlich die zugehörigen Statusinformationen mit MGET NT gelesen werden.

      Die Folge-Verarbeitung im Auftragnehmer-Vorgang beginnt bei LU6.1 erst dann wieder, wenn eine Nachricht für diesen Vorgang vorliegt. Wenn der Auftragnehmer nicht in die nächste Transaktion nach dem Rücksetzen einbezogen ist, kann dies auch erst in einer späteren Folge-Transaktion der Fall sein.

      Bei OSI TP läuft das Folge-Teilprogramm vom letzten Sicherungspunkt im Auftragnehmer-Vorgang, der PEND RS aufgerufen hat, in jedem Fall in der nächsten Transaktion: Das Programm wird gestartet, wenn eine Nachricht vom Auftraggeber eingetroffen ist oder wenn der Auftraggeber Transaktionsende angefordert hat.
  - Ging die **Ausgabenachricht an einen Vorgang**, dann startet openUTM das beim letzten Sicherungspunkt angegebene Folge-Teilprogramm. Dieses Teilprogramm muss die Rücksetznachricht lesen; anschließend kann die Ausgabenachricht gelesen werden; eventuell liegen auch noch Statusinformationen von Auftragnehmer-Vorgängen vor. War der (oberste) Auftraggeber-Vorgang an der zurückgesetzten Transaktion beteiligt, dann gibt openUTM die Meldung K034 aus.

Programmierter PEND ER/FR

Bezüglich einer verteilten Transaktion wirken PEND FR und PEND ER gleich; beim PEND FR wird jedoch - im Gegensatz zum PEND ER - kein DUMP erzeugt. Mit PEND FR kann man auf Fehler reagieren, die keine Programmfehler sind (z.B. unbrauchbare Daten).

Die Wirkung des Aufrufs PEND ER oder PEND FR ist im Auftraggeber- und im Auftragnehmer-Vorgang unterschiedlich:

- Erfolgt er im Auftraggeber-Vorgang, so werden dieser und alle Auftragnehmer-Vorgänge mit PEND ER/FR beendet.

- Erfolgt der Aufruf im Auftragnehmer-Vorgang, so wird nur dieser Vorgang beendet und die verteilte Transaktion auf den letzten Sicherungspunkt zurückgesetzt. An diesem Sicherungspunkt wird nun der Vorgangs-Wiederanlauf gestartet und das als nächstes gestartete Programm erhält die Statusinformation von demjenigen Auftragnehmer-Vorgang, der PEND ER/FR aufgerufen hat. Hat der Auftraggeber-Vorgang noch keinen Sicherungspunkt erreicht, so wird er ebenfalls beendet.

  Wurde die vorherige verteilte Transaktion im Auftragnehmer-Vorgang beendet, so wird das beim letzten Sicherungspunkt spezifizierte Folge-Teilprogramm im Auftraggeber-Vorgang gestartet.

  Will sich der Auftragnehmer-Vorgang mit PEND ER/FR beenden, so muss er vorher einen MPUT durchführen, sonst wird er beim PEND mit KCRCCC = 83Z vom System beendet. Eine Ausnahme bilden Dialoge über OSI TP, bei denen kein MPUT an den Auftraggeber erlaubt ist (KCSEND=N).
  Der Auftraggeber-Vorgang erhält immer eine Statusinformation, die mit MGET NT gelesen werden muss.

RSET

Die Wirkung des RSET-Aufrufs ist in Auftraggeber- und Auftragnehmer-Vorgängen gleich.

Das Verhalten von openUTM nach einem RSET-Aufruf in einem Teilprogrammlauf, der zu einer verteilten Transaktion gehört, ist abhängig vom Generierungsparameter RSET der UTMD-Anweisung.

- Ist RSET=LOCAL generiert, dann erlaubt openUTM, dass der RSET-Aufruf nur zum Rücksetzen der lokalen Transaktion führt. Beachten Sie bitte, dass die Datenumgebung (GSSBs, LSSBs, TLSs, ULS, ...) mit Ausnahme des SPAB auf den letzten Sicherungspunkt zurückgesetzt wird. Für innerhalb der Transaktion adressierte Auftragnehmer-Vorgänge gilt: Wenn in einem vorhergehenden, mit PEND KP oder PGWT KP beendeten Verarbeitungsschritt eine Nachricht an einen Auftragnehmer-Vorgang gesendet wurde, dann bleibt dieser Vorgang adressiert, andernfalls wird die Vorgangs-Identifikation gelöscht.
- Ist RSET = GLOBAL generiert, dann muss der Teilprogrammlauf mit PEND FR/ER/RS beendet werden. Dies führt dann auch zum Rücksetzen der verteilten Transaktion.

5.2.2.2 Fehlerbehandlung nach Vorgangs-Wiederanlauf

Musste die verteilte Transaktion zurückgesetzt werden, so wird beim Vorgangs-Wiederanlauf möglichst dasjenige Programm erneut gestartet, für welches beim Ende der letzten verteilten Transaktion eine Nachricht vorlag, bzw. für welches die nächste Eingabe vom Client bestimmt ist. Die Programmierregeln (siehe "Programmierregeln und Empfehlungen" für LU6.1, "Programmierregeln für Dialoge mit Functional Unit Commit" für OSI TP) stellen sicher, dass am Ende einer verteilten Transaktion genau eine Nachricht an den Client oder an ein Programm in einem Vorgang vorliegt.

Das Teilprogramm kann am Vorgangs-Kennzeichen im Feld KCKNZVG/ kccv_status im KB-Kopf erkennen, ob ein Vorgangs-Wiederanlauf stattgefunden hat: Nach einem Vorgangs-Wiederanlauf enthält dieses Feld den Wert "R".

Findet der Vorgangs-Wiederanlauf im Auftraggeber-Vorgang statt, dann erhält das Teilprogramm immer dann eine **Statusinformation** vom Auftragnehmer-Vorgang, wenn das Rücksetzen vom Auftragnehmer-Vorgang verursacht und dieser dadurch beendet wurde oder beendet wird. Eine solche Statusinformation wird mit **MGET NT** gelesen; sie ist eine Nachricht der Länge 0 und liefert in KCVGST/kcpcv_state und KCTAST/kcpta_state im KB-Rückgabebereich Vorgangs- und Transaktionsstatus des Auftragnehmer-Vorgangs (siehe "Programmierhilfen").

Beim Vorgangs-Wiederanlauf lassen sich nun 3 Fälle unterscheiden:

1. Beim letzten Sicherungspunkt lag eine Nachricht an den Client vor.

   Das Folge-Teilprogramm im Auftraggeber-Vorgang kann nun mit MGET die neue Eingabe des Client lesen und erhält dabei als Statusinformation den Vorgangs-Status O und den Transaktions-Status C.

   Wurde das Rücksetzen der Transaktion auf Grund eines Fehlers in einem Auftragnehmer-Vorgang verursacht und der Auftragnehmer-Vorgang dadurch beendet, so erhält man im Feld KCRPI die Vorgangs-ID desjenigen Vorgangs, der das Rücksetzen verursacht hat. Anschließend kann mit MGET NT und KCRN = Vorgangs-ID eine Statusinformation von diesem Vorgang gelesen werden.
2. Am letzten Sicherungspunkt liegt eine Nachricht von einem Auftragnehmer-Vorgang an einen Auftraggeber-Vorgang vor.

   Das Folge-Teilprogramm im Auftraggeber-Vorgang kann die Nachricht nun wie üblich mit MGET lesen und erhält dabei den Vorgangs- und Transaktions-Status von diesem Auftragnehmer-Vorgang.

   Wurde dieser Auftragnehmer-Vorgang durch einen Fehler zurückgesetzt und beendet, so erhält man nur eine entsprechende Statusinformation. Wurde das Rücksetzen von einem anderen Auftragnehmer-Vorgang verursacht, so erhält man als Statusinformation beim ersten MGET den Transaktionsstatus C. Anschließend können wie im Fall 1 noch eine oder mehrere Statusinformationen gelesen werden.

   Kann die Nachricht vom Auftragnehmer nicht gesendet werden (siehe nächsten Abschnitt), so erhält der Auftraggeber lediglich eine Statusinformation vom Auftragnehmer-Vorgang.
3. Beim letzten Sicherungspunkt lag eine Nachricht von einem Auftraggeber-Vorgang an einen Auftragnehmer-Vorgang vor. Wenn möglich, wird dann das Folge-Teilprogramm im Auftragnehmer-Vorgang gestartet.

   Kann das Folge-Teilprogramm nicht gestartet werden (z.B. weil die Anwendung beendet und innerhalb der generierten Wartezeit nicht wieder gestartet wurde oder weil sich der Vorgang mit PEND ER beendet hat), so wird das Folge-Teilprogramm im Auftraggeber-Vorgang gestartet und erhält eine Statusinformation.

Statusinformationen liegen von allen Auftragnehmer-Vorgängen vor, die das Rücksetzen der verteilten Transaktion verursacht haben und beendet wurden bzw. noch werden.

Wird nach einem Vorgangs-Wiederanlauf des Auftraggeber-Vorgangs wieder ein Auftragnehmer-Vorgang adressiert und tritt wieder ein Fehler auf, so kann der Auftraggeber-Vorgang mehrfach auf den gleichen Sicherungspunkt zurückgesetzt werden. Da die Statusinformationen vom vorhergehenden Rücksetzen erhalten bleiben, kann man eventuell mehrere Statusinformationen erhalten.

Liegen mehrere Statusinformationen vor, so erhält man jeweils beim MGET die Vorgangs-Identifikation eines nächsten Vorgangs mit Statusinformation. Statusinformationen von mehreren Auftragnehmer-Vorgängen müssen in der von openUTM vorgeschlagenen Reihenfolge (KCRPI) gelesen werden. Das Feld KCMF/kcfn versorgen Sie beim Lesen der Statusinformation mit Leerzeichen.

Ist ein entfernter Vorgang in der zurückgesetzten Transaktion durch APRO-Aufruf neu entstanden, so liegt von diesem möglicherweise eine Statusinformation vor, obwohl es auf dem Sicherungspunkt, auf dem neu aufgesetzt wird, diesen Vorgang eigentlich noch nicht gibt.

Keine Nachricht vom Auftragnehmer

Es gibt zwei Möglichkeiten, weshalb ein Auftragnehmer-Vorgang kein Ergebnis an den Auftraggeber-Vorgang senden kann.

1. der Auftragnehmer-Vorgang wurde aus einem der folgenden Gründe nicht gestartet:
   - es existiert keine logische Verbindung zur Anwendung des Auftragnehmers und innerhalb der generierten Wartezeit konnte keine Verbindung aufgebaut werden.
   - innerhalb der generierten Wartezeit konnte keine Session bzw. Association zur Anwendung des Auftragnehmers belegt werden.
   - der Auftrag wurde zur Anwendung des Auftragnehmers gesendet, aber der übermittelte Transaktionscode ist unbekannt oder gesperrt.
   - die Anwendung des Auftraggebers wird mit KDCSHUT W beendet.
2. Der Auftragnehmer-Vorgang wurde gestartet, aber bei der Bearbeitung des Auftragnehmer-Vorgangs traten Fehler auf oder der Kommunikationsweg wurde gestört. Deshalb wurde die Transaktion im Auftragnehmer-Vorgang zurückgesetzt und der Auftragnehmer-Vorgang wurde beendet. Außerdem wurde die Session /Association zum fehlerhaft beendeten Auftragnehmer-Vorgang freigegeben. Es können folgende Fehlersituationen auftreten:
   - innerhalb der generierten Wartezeit ist keine Antwort vom Auftragnehmer-Vorgang bei der Anwendung des Auftraggebers eingetroffen.
   - die Anwendung des Auftragnehmers wurde wegen eines gravierenden Fehlers abnormal beendet.
   - der Auftragnehmer-Vorgang hat sich mit PEND ER beendet oder wurde mit gravierendem Programmfehler beendet.

5.2.3 Lastverteilung über LPAP-Bündel

openUTM bietet die Funktion der LPAP-Bündel für das OSI TP- und das LU6.1-Protokoll.

LPAP-Bündel ermöglichen die Lastverteilung bzw. die Nutzung von alternativen Verbindungen zu einer Partner-Anwendung. Soll eine UTM-Anwendung sehr viele Nachrichten mit einer Partner-Anwendung austauschen, kann es für die Lastverteilung sinnvoll sein, mehrere Instanzen der Partner-Anwendung zu starten und die Nachrichten auf die einzelnen Instanzen zu verteilen. In einem LPAP-Bündel übernimmt openUTM die Verteilung der Nachrichten an die Instanzen der Partner-Anwendung. Ein LPAP-Bündel besteht aus einem Master-LPAP und mehreren Slave-LPAPs.

Die Slave-LPAPs werden dem Master-LPAP bei der Generierung zugeordnet. Die physikalischen Verbindungen (CONs) der einzelnen Slave-LPAPs adressieren dabei (im Normalfall) unterschiedliche Partner-Anwendungen.

Anwendung betreiben

Damit openUTM die Nachrichten an die Slave-LPAPs verteilt, richten Sie in den Teilprogrammen die Nachrichten an das Master-LPAP.

openUTM verteilt diese Nachrichten reihum an die Slave-LPAPs. Dabei wird immer versucht ein Slave-LPAP zu finden, über das die Nachricht auch gesendet werden kann, d.h. zu dem bereits eine Verbindung aufgebaut ist und für das z.B. der Queue-Level noch nicht überschritten ist.

Weitere Details dazu entnehmen Sie der Beschreibung des APRO-Aufrufs ab "APRO Adressieren eines Auftragnehmer-Vorgangs".

Administration

Über die Administrationsschnittstelle werden die Eigenschaften "Master-LPAP" und "Slave-LPAP" angezeigt.

Zu einem Master-LPAP werden alle Slave-LPAPs angezeigt, zu einem Slave-LPAP das Master-LPAP.

Für Master-LPAPs können Sie administrativ den Status auf ON oder OFF setzen. Wenn Sie den Status eines Master-LPAPs ändern, wird damit auch der Status an allen Slave-LPAPs entsprechend geändert.

5.3 Verteilte Dialoge über LU6.1

Das LU6.1-Protokoll (**L**ogical **U**nit **6.1**) ist ein von IBM definiertes SNA-Protokoll. Es wurde laufend erweitert und hat sich zu einem Industriestandard entwickelt. Die Kommunikation erfolgt mit anwendungsübergreifender Transaktionssicherung.

LU6.1 eignet sich nicht nur für die Kopplung von UTM-Anwendungen, sondern auch für die Kopplung von UTM-Anwendungen mit CICS- und IMS-Anwendungen, die auf IBM-Rechnern laufen.

In den nächsten Abschnitten erfahren Sie, welche Programmierhilfen openUTM für verteilte LU6.1-Dialoge zur Verfügung stellt, welche Regeln für solche Dialoge gelten und wie Sie bestehende Teilprogramme als LU6.1-Auftragnehmer verwenden können.

5.3.1 Programmierhilfen

openUTM stellt den Teilprogrammen eine Reihe von Informationen zur Verfügung, die ausgewertet und für die Steuerung der Kommunikation verwendet werden können:

- nach dem INIT-Aufruf
- nach dem Lesen der Dialog-Nachricht mit MGET

Nach dem INIT-Aufruf kann man u.a. aus dem Kommunikationsbereich entnehmen,

- unter welcher Benutzerkennung bzw. unter welchem Session-Namen das Teilprogramm gestartet wurde (Feld KCBENID/kcuserid)
- welches Kommunikationsprotokoll der Partner verwendet (Feld KCCP, bei LU6.1 ist hier 1 eingetragen)
- ob es sich um einen Vorgangs-Wiederanlauf handelt (KCKNZVG/kccv_status enthält dann den Wert "R").

Nach dem MGET-Aufruf gibt openUTM im Feld KCVGST/kcpcv_state den Status des Partner-Vorgangs und im Feld KCTAST/kcpta_state den Status der Partner-Transaktion zurück.

Für COBOL ist ein übergeordnetes Statusfeld KCRST definiert, das die Felder KCVGST und KCTAST enthält.

Mit Hilfe dieser Stati lässt sich z.B. feststellen, ob der Partner-Vorgang bereits Transaktionsende angefordert hat und auf die Beendigung der verteilten Verarbeitung wartet. Dadurch kann einerseits der Programmlauf so gesteuert werden, dass die Einhaltung der Programmierregeln auch ohne genaue Kenntnis des Sicherungsverhaltens des Partners garantiert werden kann, andererseits besteht auch die Möglichkeit, im Auftraggeber-Vorgang auf Fehler im Auftragnehmer-Vorgang zu reagieren. Die möglichen Stati entnehmen Sie den beiden folgenden Tabellen.

Vorgangs-Status des Partners

KCVGST/ kcpcv_state	Bedeutung
I	(inactive): Der Vorgang ist inaktiv, d.h. er ist noch nicht gestartet worden oder er konnte nicht gestartet werden, z.B. weil der TAC in der Partner-Anwendung unbekannt ist.
O	(open): Der Vorgang ist offen.
C	(closed): Der Vorgang hat sich mit PEND FI beendet.
R	(reset): Der Vorgang wurde vom Programm mit PEND RS beendet.
E	(error): Der Vorgang wurde vom Programm mit PEND ER/FR beendet.
Z	der Vorgang wurde von openUTM wegen eines Fehlers beendet.

KCVGST/ kcpcv_state	Bedeutung
T	(timeout): Der Vorgang wurde bzw. wird fehlerhaft beendet, da innerhalb der generierten Wartezeit keine Antwort eingetroffen ist, oder da innerhalb der generierten Wartezeit keine freie Session belegt werden konnte.

Transaktionsstatus des Partners

KCTAST/ kcpta_state	Bedeutung
I	(inactive): Es existiert keine Transaktion, da auch kein Vorgang existiert.
O	(open): Die Transaktion ist offen; der letzte Verarbeitungsschritt wurde mit PEND KP beendet.
P	(prepare to commit): Der Vorgang hat Transaktionsende angefordert, die verteilte Transaktion ist jedoch noch nicht beendet. In dieser Situation wartet der Vorgang auf eine Quittung, die er erhält, sobald die verteilte Transaktion beendet ist. Solange bleiben, ähnlich wie beim PEND KP, Betriebsmittel gesperrt (GSSB, TLS).
C	(closed): Die letzte verteilte Transaktion, an der der Vorgang beteiligt war, ist beendet.
R	(reset): Die verteilte Transaktion und damit auch die lokale Transaktion wurde zurückgesetzt.
M	(mismatch): Die an einer verteilten Transaktion beteiligten Vorgänge können sich beim Vorgangswiederanlauf nicht auf einen gemeinsamen Sicherungspunkt einigen. Dies kann nur bei einem Timeout oder nach Beendigung und Start einer UTM-F-Anwendung auftreten.

5.3.2 Programmierregeln und Empfehlungen

In den folgenden Abschnitten werden die Regeln genannt, die für verteilte Dialoge über LU6.1 einzuhalten sind. Wenn Sie die im abschließenden Abschnitt genannte "Programmierempfehlungen" beachten, werden die genannten Regeln automatisch eingehalten.

> ¡ Bitte beachten Sie, dass die Aufrufe PGWT CM und PGWT RB in verteilten Dialogen über LU6.1 nicht erlaubt sind.

Wirkung der PEND-Aufrufe und Einsatzregeln

Die PEND-Aufrufe in Auftraggeber- **und** Auftragnehmer-Vorgang steuern die verteilte Transaktion, d.h. sie entscheiden, wann der **gemeinsame** Sicherungspunkt der beiden Transaktionen gesetzt wird.

Welche PEND-Aufrufe ein Vorgang verwenden darf und welche Wirkung diese haben, ist abhängig von Vorgangs- und Transaktions-Status des Partner-Vorgangs.
In der folgenden Tabelle ist die Wirkung der PEND-Aufrufe im Auftragnehmer- und im Auftraggeber-Vorgang beschrieben sowie die Regeln, die für deren Einsatz gelten:

Variante	Wirkung/Einsatzregeln
PEND KP PGWT KP	Es wird kein Sicherungspunkt angefordert und kein Sicherungspunkt gesetzt. Der Vorgang bleibt im Transaktionsstatus "O" und im Vorgangsstatus "O". Eine offene DB-Transaktion bleibt offen. Im Auftraggeber-Vorgang wird das in KCRN angegebene Folge-Teilprogramm gestartet, sobald alle Ergebnisse von den Auftragnehmer-Vorgängen eingetroffen sind. Zulässig wenn das Teilprogramm eine Nachricht an das Terminal oder den Client (LTERM-Partner) sendet oder wenn die Nachricht an einen Partner-Vorgang gerichtet ist und dieser sich nicht im Transaktionsstatus "P" befindet.
PEND RE	Es wird Transaktionsende (Sicherungspunkt) angefordert. Befinden sich alle Partner-Vorgänge bereits im Transaktionsstatus "P" (d.h. sie haben schon Transaktionsende angefordert), so wird von den beteiligten Vorgängen das Transaktionsende durchgeführt, d.h. es wird ein gemeinsamer Sicherungspunkt gesetzt. Befindet sich ein Partner-Vorgang nicht im Transaktionsstatus "P", so geht der lokale Vorgang in den Transaktionsstatus "P" und wartet, dass der Partner-Vorgang ebenfalls Transaktionsende anfordert. Im Auftraggeber-Vorgang wird das in KCRN angegebene Folge-Teilprogramm gestartet, sobald alle Ergebnisse von den Auftragnehmer-Vorgängen eingetroffen sind. Zulässig in folgenden beiden Fällen: • es gibt keinen Partner-Vorgang mit offener Transaktion • es gibt genau einen Partner-Vorgang mit offener Transaktion und im Teilprogramm wird eine Nachricht an diesen Partner gesendet. Wenn es mehrere Partner-Vorgänge mit offenen Transaktionen gibt ist PEND RE also verboten.

Variante	Wirkung/Einsatzregeln
PEND FI	Die Wirkung ist die gleiche wie beim PEND RE, nur dass gleichzeitig Vorgangsende angefordert wird. Der Vorgang wird beim nächsten Sicherungspunkt beendet. Zulässig, wenn kein Auftragnehmer-Vorgang mehr offen ist. D.h. alle Auftragnehmer müssen PEND FI abgesetzt haben, bevor der Auftraggeber PEND FI absetzen darf.
PEND FC	Die Wirkung ist die gleiche wie beim PEND FI, nur dass nach Vorgangsende sofort der per Vorgangs-Kettung angeschlossene Folge-Vorgang gestartet wird. Zulässig nur in Auftraggeber-Vorgängen (in Auftragnehmern also grundsätzlich verboten) und nur dann, wenn kein Auftragnehmer-Vorgang mehr offen ist. D.h. alle Auftragnehmer müssen PEND FI abgesetzt haben, bevor der Auftraggeber PEND FC absetzen darf.
PEND PR PEND PA	kein Einfluss auf Transaktion.
PEND SP	Die Transaktion wird geschlossen, d.h. ein Sicherungspunkt wird gesetzt und der Verarbeitungsschritt wird fortgesetzt. Zulässig, wenn es keinen Partner-Vorgang mit offener Transaktion gibt.
PEND RS	Mit PEND RS werden alle an der verteilten Transaktion beteiligten Vorgänge auf den letzten Sicherungspunkt zurückgesetzt. Alle Vorgänge, die in der verteilten Transaktion entstanden sind, werden durch das Rücksetzen beendet. Vorgänge, die mindestens einen Sicherungspunkt erreicht haben, bleiben offen.

Programmierregeln

Beim Programmieren verteilter Transaktionen sind eine Reihe von Regeln einzuhalten, deren Verletzung den Abbruch des betroffenen Vorgangs mit (internem) PEND ER und KCRCCC=87Z zur Folge hat.

Diese Regeln bestimmen:

- wie Transaktionen und Vorgänge bei verteilter Verarbeitung beendet werden müssen und
- an wen die Ausgabenachricht gesendet werden darf.

Das Beenden von Teilprogrammläufen, Transaktionen und Vorgängen wird wie gewohnt über die verschiedenen PEND-Varianten realisiert; wichtig sind dabei die PEND-Varianten KP, RE, SP, FI, FC und für die Fehlerbehandlung RS, ER und FR.

Bei den Varianten KP, RE und SP wird in KCRN das Folge-Teilprogramm angegeben, mit dem der lokale Vorgang fortgesetzt wird. Dieses Folge-Teilprogramm wird bei PEND KP/RE gestartet, sobald alle Ergebnisse von den Auftragnehmern eingetroffen sind, oder - falls die Nachricht an den Auftraggeber gerichtet war - sobald die nächste Nachricht vom Auftraggeber eintrifft. Bei PEND SP wird das Folge-Teilprogramm sofort gestartet.

Der Aufruf PGWT KP kann immer dann verwendet werden, wenn auch ein PEND KP erlaubt ist.

Für die korrekte Programmierung einer verteilten Transaktion gelten folgende Regeln:

- **Vorgangs-Regel:** Ein Auftraggeber-Vorgang darf erst beendet werden, nachdem alle zugehörigen Auftragnehmer-Vorgänge (mit PEND FI) beendet wurden. PEND FC ist in Auftragnehmer-Vorgängen verboten. Der Auftraggeber-Vorgang kann nach dem PEND FI des Auftragnehmer-Vorgangs den folgenden Verarbeitungsschritt mit PEND KP, RE, SP, FI oder FC beenden. Er darf keine Nachricht mehr an diesen Auftragnehmer-Vorgang senden.

- **Transaktionsregel (einstufig):** Ein Auftraggeber- oder Auftragnehmer-Vorgang kann einen Verarbeitungsschritt wahlweise mit oder ohne Transaktionsende abschließen, wobei folgende Einschränkungen beachtet werden müssen:
  - Ein Auftraggeber-Vorgang darf **kein Transaktionsende** anfordern, wenn im Auftragnehmer-Vorgang die Transaktion offen ist und die Ausgabenachricht an den Client gerichtet ist.
  - Ein Vorgang **muss Transaktionsende** anfordern, wenn die Ausgabenachricht an einen Partner-Vorgang gerichtet ist, der Transaktionsende angefordert hat.

Bei mehrstufiger verteilter Transaktion, d.h. wenn ein Auftragnehmer-Vorgang selbst wieder Auftraggeber-Vorgang ist oder wenn ein Auftraggeber-Vorgang in einer Transaktion mehrere Auftragnehmer-Vorgänge adressiert, dann wird die Transaktionsregel wie folgt verallgemeinert:

- **Transaktionsregel (mehrstufig)**: Ein Vorgang kann einen Verarbeitungsschritt wahlweise mit oder ohne Transaktionsende abschließen, wobei folgende Einschränkungen beachtet werden müssen:
  - Ein Vorgang darf **kein Transaktionsende** anfordern, wenn es einen Partner-Vorgang mit offener Transaktion gibt und die Ausgabenachricht nicht an diesen Partner-Vorgang gerichtet ist. Wenn es mehrere Partner-Vorgänge mit offenen Transaktionen gibt, ist es somit grundsätzlich verboten Transaktionsende anzufordern.
  - Ein Vorgang **muss Transaktionsende** anfordern, wenn die Ausgabenachricht an einen Partner-Vorgang gerichtet ist, der Transaktionsende angefordert hat.

Hat ein Vorgang mehrere Partner-Vorgänge mit offener Transaktion, dann sind PEND RE, PEND FI und PEND SP grundsätzlich nicht erlaubt.

In welcher Situation welcher PEND-Aufruf zulässig ist, zeigen die Tabellen „Wirkung der PEND-Aufrufe und Einsatzregeln", „PEND-Varianten im Auftraggeber" und „PEND-Varianten im Auftragnehmer-Vorgang".

Befolgt man die nachfolgend beschriebene Bottom-Up-Strategie, dann werden die Programmierregeln automatisch eingehalten.

Programmierempfehlung: Bottom-Up-Strategie

Die Bottom-Up-Strategie besteht darin, dass verteilte Transaktionen immer von unten nach oben beendet werden, d.h.:

- Ein Auftragnehmer-Vorgang fordert immer vor seinem Auftraggeber-Vorgang Transaktionsende an und sendet seine Ausgabenachricht an seinen Auftraggeber-Vorgang.
- Ein Auftraggeber-Vorgang fordert erst Transaktionsende an, wenn alle seine Auftragnehmer-Vorgänge Transaktionsende angefordert haben. Die Ausgabenachricht geht wiederum an seinen eigenen Auftraggeber.

PEND-Varianten in Abhängigkeit vom Partner-Status

Beim PEND-Aufruf muss immer auf den Vorgangs-Status und den Transaktions-Status des Partner-Vorgangs geachtet werden. Diese Statusangaben gibt openUTM nach dem MGET-Aufruf in den Feldern KCVGST /kcpcv_state und KCTAST/kcpta_state zurück.

Bei den in folgenden Tabellen dargestellten Fällen werden keine neuen Regeln eingeführt, sie illustrieren lediglich die in den vorhergehenden Abschnitten genannten Regeln.

PEND-Varianten im Auftraggeber

Die Aufrufe PEND PA und PEND PR sind nicht berücksichtigt, da es bei ihnen keine Besonderheiten bei verteilter Verarbeitung gibt.

Partnerstatus		**PEND-Varianten im Auftraggeber-Vorgang**	
KCVGST/ kcpcv_state	**KCTAST/ kcpta_state**	**erlaubte Varianten und deren Wirkung und Einschränkungen**	
"O"	"O"	KP:	Transaktionen im Auftraggeber- und Auftragnehmer-Vorgang bleiben offen - Normalfall.
		RE:	fordert Transaktionsende an. Nur erlaubt, wenn die Ausgabenachricht an diesen Auftragnehmer gerichtet ist und es keinen weiteren Partner mit offener Transaktion gibt.
		RS:	setzt Auftraggeber- und Auftragnehmer-Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	beendet Auftraggeber- und Auftragnehmer-Vorgang und setzt die verteilte Transaktion zurück.
"O"	"P"	KP:	Nur erlaubt, wenn die Ausgabenachricht an einen anderen Partner-Vorgang oder an den Client gerichtet ist; die Transaktionen bleiben offen (nicht zu empfehlen wegen PTC-Zustand)
		RE:	Die verteilte Transaktion wird beendet.
		SP:	Die verteilte Transaktion wird beendet.
		RS:	setzt Auftraggeber- und Auftragnehmer-Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	beendet Auftraggeber- und Auftragnehmer-Vorgang und setzt die verteilte Transaktion zurück.

Partnerstatus		PEND-Varianten im Auftraggeber-Vorgang	
KCVGST/ kcpcv_state	**KCTAST/ kcpta_state**	**erlaubte Varianten und deren Wirkung und Einschränkungen**	
"O"	"C"	KP:	Die Transaktion im Auftraggeber-Vorgang bleibt offen.
		RE:	Die Transaktion im Auftraggeber-Vorgang wird beendet, wenn die Ausgabe-Nachricht an den Client gerichtet ist, andernfalls geht der Vorgang in den Transaktionsstatus "P".
		SP:	Die Transaktion im Auftraggeber-Vorgang wird beendet.
		RS:	setzt die Auftraggeber-Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	beendet Auftraggeber- und Auftragnehmer-Vorgang und setzt die lokale Transaktion zurück.
"C"	"P"		Die Ausgabenachricht muss an einen anderen Partner oder an den Client oder (bei PEND SP/FC) an ein Folge-Teilprogramm gehen!
		KP:	nicht zu empfehlen, da der Auftragnehmer-Vorgang im Zustand PTC wartet.
		RE:	beendet die verteilte Transaktion und den Auftragnehmer-Vorgang
		SP:	wie bei RE
		FI:	beendet die verteilte Transaktion sowie Auftraggeber-und Auftragnehmer-Vorgang
		FC:	wie bei FI
		RS:	setzt Auftraggeber- und Auftragnehmer-Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	beendet Auftraggeber- und Auftragnehmer-Vorgang und setzt die verteilte Transaktion zurück.

Partnerstatus		PEND-Varianten im Auftraggeber-Vorgang	
KCVGST/ kcpcv_state	**KCTAST/ kcpta_state**	**erlaubte Varianten und deren Wirkung und Einschränkungen**	
"C"	"C"		Die Ausgabenachricht muss an einen anderen Partner oder an den Client oder (bei PEND SP/FC) an ein Folge-Teilprogramm gehen!
		KP:	Die Transaktion im Auftraggeber bleibt offen.
		RE:	Die Transaktion im Auftraggeber wird beendet
		SP:	Die Transaktion im Auftraggeber wird beendet
		FI:	Transaktion und Auftraggeber-Vorgang werden beendet.
		FC:	Transaktion und Auftraggeber-Vorgang werden beendet.
		RS:	setzt die Auftraggeber-Transaktion zurück auf den letzten Sicherungspunkt.
		ER /FR:	setzt die Transaktion im Auftraggeber-Vorgang zurück und beendet den Auftraggeber-Vorgang. Der Auftragnehmer-Vorgang ist bereits beendet.

Die Kombinationen "KCVGST=O, KCTAST=C" und "KCVGST=C, KCTAST=C" können nicht auftreten, wenn die "Bottom-Up-Strategie" eingehalten wird.

PEND-Varianten im Auftragnehmer-Vorgang

Die Aufrufe PEND PA und PEND PR sind dabei nicht berücksichtigt, da es bei ihnen keine Besonderheiten bei verteilter Verarbeitung gibt. Die Variante PEND FC ist in Auftragnehmer-Vorgängen nicht erlaubt.

Partnerstatus		**PEND-Varianten im Auftragnehmer-Vorgang**	
KCVGST/ kcpcv_state	**KCTAST/ kcpta_state**	**erlaubte Varianten und deren Wirkung und Einschränkungen**	
"O"	"O"	KP:	Die Transaktionen in Auftragnehmer- und Auftraggeber-Vorgang bleiben offen.
		RE:	Der Auftragnehmer-Vorgang geht in den Transaktionsstatus P.
		FI:	Der Auftragnehmer-Vorgang geht in den Transaktionsstatus P. Der Vorgang wird beim nächsten Sicherungspunkt (=Ende der verteilten Transaktion) beendet.
		RS:	setzt die verteilte Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	Die verteilte Transaktion wird zurückgesetzt; der Auftragnehmer-Vorgang wird beendet.
"O"	"P"	RE:	Die verteilte Transaktion wird beendet.
		SP:	Die verteilte Transaktion wird beendet.
		FI:	Die verteilte Transaktion und der Auftragnehmer-Vorgang werden beendet.
		RS:	setzt die verteilte Transaktion zurück auf den letzten Sicherungspunkt. Vorgänge, die in dieser Transaktion entstanden sind, werden beendet.
		ER /FR:	Die verteilte Transaktion wird zurückgesetzt; der Auftragnehmer-Vorgang wird beendet.
"O"	"C"	KP:	erlaubt
		RE:	Der Auftragnehmer-Vorgang geht in den Transaktionsstatus P.
		SP:	Der Auftragnehmer-Vorgang beendet eine lokale Transaktion
		FI:	Der Auftragnehmer-Vorgang geht in den Transaktionsstatus P. Der Vorgang wird beim nächsten Sicherungspunkt (=Ende der verteilten Transaktion) beendet.
		RS:	setzt die verteilte Transaktion zurück auf den letzten Sicherungspunkt.
		ER /FR:	Die verteilte Transaktion wird zurückgesetzt; der Auftragnehmer-Vorgang wird beendet.

Wenn die Bottom-Up-Strategie eingehalten wird, dann kann nur die Kombination "O"/"O" auftreten, siehe "Bottom-Up-Strategie".

Vor einem PEND RS in einer Folge-Transaktion müssen Sie mit MPUT RM eine Rücksetznachricht schicken, sonst bricht openUTM den Auftragnehmer-Vorgang mit 83Z ab.

5.3.3 Bestehende Teilprogramme als LU6.1-Auftragnehmer

UTM-Teilprogramme, die für die Kommunikation mit Terminals konzipiert wurden, lassen sich unter bestimmten Rahmenbedingungen unverändert als Teilprogramme eines Auftragnehmer-Vorgangs einsetzen (siehe nachfolgende Liste). Auch bestehende Asynchron-Programme müssen für die Verwendung als Auftragnehmer nicht angepasst werden.

Ein und derselbe Service kann also sowohl von Terminals und Client-Programmen als auch von anderen Vorgängen genutzt werden. Dadurch ermöglicht openUTM hohe Flexibilität bei der Verteilung von Anwendungen.

Wenn Sie bestehende Teilprogramme unverändert als Auftragnehmer-Teilprogramme verwenden wollen, oder Teilprogramme entwickeln wollen, die sowohl von Terminals und Client-Programmen als auch von anderen Vorgängen genutzt werden können, müssen Sie folgende Punkte beachten:

- Unterschiedliche Rückgaben im KB-Kopf

  Der Kommunikationspartner des Auftragnehmer-Vorgangs ist der Auftraggeber-Vorgang, nicht der Benutzer am Terminal. Daher bekommt der Auftragnehmer-Vorgang im KB-Kopf weder die Kennung des Terminal-Benutzers noch den Namen des LTERM-Partners (beide sind in der Anwendung des Auftragnehmer-Vorgangs möglicherweise gar nicht generiert). Er erhält statt dessen den lokalen Session-Namen (LSES-Namen) und den lokalen Namen der Partner-Anwendung (LPAP-Namen).

- Unterschiedliche Zuordnung von TLS und ULS

  Schreib- und Leseaufrufe für TLS beziehen sich im Auftragnehmer-Vorgang auf den TLS des LPAP-Partners und nicht auf einen TLS eines LTERM-Partners; desgleichen beziehen sich Aufrufe für einen ULS auf den ULS der Session.

- Funktionstasten bei verteilter Verarbeitung nicht nutzbar

  Der Auftraggeber-Vorgang kann dem Auftragnehmer-Vorgang keine den Funktionstasten entsprechende Nachricht senden. Der Auftragnehmer-Vorgang kann deshalb beim MGET-Aufruf nie den entsprechenden KDCS-Returncodes (19Z bis 39Z) erhalten.

- Ausweisleser im Auftragnehmer-Vorgang nicht nutzbar

  In einem Auftragnehmer-Vorgang enthält das Feld KCAUSWEIS/kccard immer Leerzeichen.

- Keine Formatierung bei verteilter Verarbeitung über LU6.1

  Für einen Vorgang ist es im Allgemeinen unerheblich, ob er die Dialog-Nachricht von einem Terminal, von einem UTM-Client-Programm oder von einem LU6.1-Partner erhält. openUTM überträgt bei verteilter Verarbeitung über LU6.1 zwar das Formatkennzeichen, das im Auftraggeber-Vorgang beim MPUT-Aufruf angegeben wurde, formatiert allerdings nicht. Das Formatkennzeichen wird auch bei allen Teilnachrichten übertragen.

  Falls beim MGET ein falsches Formatkennzeichen angegeben ist, verhält sich openUTM bei verteilter Verarbeitung über LU6.1 wie beim Einsatz von Teilformaten bei Terminals: openUTM quittiert ein falsches Formatkennzeichen beim MGET mit dem KDCS-Returncode 03Z und es wird keine (Teil-)Nachricht in den Nachrichtenbereich übertragen.

- Spezieller Aufbau des Auftraggeber-Vorgang (bei verteilten Dialogen)

  Wenn bereits bestehende Teilprogramme einer Anwendung als Teilprogramme in einem Auftragnehmer-Vorgang verwendet werden sollen oder wenn die Auftragnehmer-Programme so programmiert werden, dass sie die Statusanzeigen beim MGET-Aufruf nicht auswerten, so muss sich der Auftraggeber-Vorgang bezüglich der Transaktionssicherung vom Auftragnehmer-Vorgang steuern lassen. Das heißt, die Bottom-Up-Strategie (siehe "Programmierregeln und Empfehlungen") muss eingehalten werden. Der Auftraggeber-Vorgang muss hierzu den Transaktionsstatus des Auftragnehmers beachten: Das Transaktionsende (PEND RE/SP) darf im Auftraggeber erst dann gesetzt werden, wenn sich alle Auftragnehmer im Transaktionsstatus "P" befinden.

5.3.4 Beispiel: verteilter Dialog über LU6.1

Anhand eines einfachen Beispiels wird im Folgenden der Ablauf der Aufrufe bei verteilten Dialogen über LU6.1 dargestellt. Dabei ist jeweils angegeben, welche Felder versorgt oder ausgewertet werden können bzw. müssen. Die Feld-Namen sind in der COBOL-Notation angegeben.

Das Auftraggeber-Programm besteht in diesem Beispiel aus zwei Teilprogrammen: im ersten Teil wird der Unterauftrag an den Auftragnehmer-Vorgang gegeben, im zweiten Teil wird die Antwort des Auftragnehmer-Vorgangs gelesen und ggf. die Antwort an das Terminal ausgegeben.

Wie in Abschnitt „Fehlerbehandlung im Teilprogramm“ erläutert, wird ein Auftraggeber-Teilprogramm, das auf einen Sicherungspunkt folgt, nicht nur beim normalen Ablauf gestartet, sondern auch nach Rücksetzen einer verteilten Transaktion beim Vorgangs-Wiederanlauf. Aus diesem Grund wird im ersten Teil des Auftraggeber-Programms geprüft, ob eine Statusinformation vorliegt. Ein solcher Vorgangs-Wiederanlauf findet aber nur statt, wenn der Auftraggeber-Vorgang vor dem Start dieses Teilprogramms einen Sicherungspunkt gesetzt hat (mit oder ohne Beteiligung eines Auftragnehmer-Vorgangs).

1. Auftraggeber-Programm, erster Teil

```
INIT
       Auswerten:
       KCKNZVG     --> R   es handelt sich um einen Vorgangs-Wiederanlauf,
                           eventuell liegt eine Statusinformation vor.
MGET   Eingabenachricht vom Terminal. Hat der Auftraggeber-Vorgang bereits einen
       Sicherungspunkt erreicht, so kann zusätzlich eine Statusinformation vorliegen.
       Auswerten:
       KCRPI       -->  Rückgabeinformation vom Auftragnehmer
                 Leerzeichen    es liegt keine Statusinformation vor
                 >vgid     es liegt eine Statusinformation vom Auftragnehmer-Vorgang mit der
                           angegebenen Vorgangs-Identifikation vor. In diesem Fall ist die
                           Statusinformation mit einem 2. MGET zu lesen.
2. MGET
       Lesen der Statusinformation, danach Fehlerbehandlung notwendig.
       Versorgen:
       KCOM        <-- mit NT
       KCLA        <-- mit der Länge 0
       KCRN        <-- mit der Vorgangs-Identifikation (>vgid)
       KCMF        <-- mit Leerzeichen
       Auswerten:
       KCVGST      -->  Vorgangs-Status:
                 I         Auftragnehmer-Vorgang inaktiv
                 E         Auftragnehmer-Vorgang hat sich mit PEND ER/FR beendet
                 Z         Auftragnehmer-Vorgang wurde von openUTM mit PEND ER beendet
                 R         Auftragnehmer-Vorgang hat sich mit PEND RS beendet
                 T         Zeit ist abgelaufen (Timer)
       KCTAST      -->  Transaktionsstatus:
                 I         Transaktion in Auftragnehmer-Vorgang inaktiv
                 R         Transaktion in Auftragnehmer-Vorgang zurückgesetzt
                 M         Mismatch
Wenn keine Statusinformation vorliegt:
APRO   Adressieren des Auftragnehmer-Vorgangs (wenn nicht schon geschehen)
       Versorgen:
       KCOM        <-- mit DM für Dialog-Vorgang
       KCRN        <-- mit dem LTAC des Auftragnehmer-Vorgangs
       KCPA        <-- bei zweistufiger Adressierung:
```

```
                        mit dem Namen der Partner-Anwendung
       KCPI        <-- mit einer selbstgewählten Vorgangs-Identifikation (>vgid)
       KCLM        <-- 0
MPUT   an Auftragnehmer-Vorgang
       Versorgen:
       KCOM        <-- NT oder NE
       KCRN        <-- mit der Vorgangs-Identifikation (>vgid)
       KCMF        <-- evtl. Formatkennzeichen für Auftragnehmer
       KCDF        <-- binär null
       KCLM        <-- Länge
PEND   (Ende erster Teil)
       Versorgen:
       KCOM        <-- Ausprägung des PEND-Aufrufs:
                 KP        im Normalfall im Auftraggeber-Vorgang zu empfehlen
                 RE        Transaktionsende wird angefordert
                 FI        nicht erlaubt, weil noch Auftragnehmer offen
                 ER/FR     bricht auch Auftragnehmer-Vorgang ab!
                 PA        nach MPUT an Auftragnehmer-Vorgang verboten
                 PR        nach MPUT an Auftragnehmer-Vorgang verboten
                 SP        nach MPUT an Auftragnehmer-Vorgang verboten
                 FC        nach MPUT an Auftragnehmer-Vorgang verboten
       KCRN        <-- Name des Folge-Teilprogramms des Auftraggebers
                       (zweiter Teil des Auftraggeber-Programms)
```

2. Auftragnehmer-Programm

```
INIT
      Auswerten:
      KCBENID      --> Name der Session
      KCLOGTER     --> Name der Partner-Anwendung
      KCTERMN      --> Kennzeichen der Partner-Anwendung
      KCCP         --> Kennzeichen für das verwendete Protokoll, bei LU6.1 wird '1'
                       eingetragen
      KCRMF        --> Formatkennzeichen aus erstem MPUT des Auftraggebers
MGET  Nachricht von Auftraggeber-Vorgang lesen
      Auswerten:
      KCRCCC       --> KDCS-Fehlercode, 19Z bis 39Z können nicht auftreten
      KCRLM        --> Länge aus MPUT des Auftraggebers (KCLM)
      KCRST 1.Byte --> Vorgangs-Status des Auftraggeber-Vorgangs:
                       O   Auftraggeber-Vorgang ist offen
      KCRST 2.Byte --> Transaktions-Status des Auftraggeber-Vorgangs
                       O   Transaktion ist offen (PEND KP bei Auftraggeber)
                       P   Transaktionsende eingeleitet (PEND RE)
                       C   Transaktion bei Auftraggeber beendet
      KCRMF            wenn noch Folgeteile vorliegen: das Formatkennzeichen des nächsten
                                                       Nachrichtenteils,
                       sonst: das Formatkennzeichen des gelesenen Teils.
MPUT  unverändert
      Versorgen:
      KCOM         <-- mit NT oder NE
      KCMF         <-- Formatkennzeichen oder Leerzeichen
      KCLM         <-- Länge der Nachricht
      KCRN         <-- mit Leerzeichen, um die Nachricht an den Auftraggeber zu senden
      KCDF         <-- beliebiger Wert, den der Auftraggeber beim MGET erhält
PEND  Ende Auftragnehmer-Teilprogramm
      Versorgen:
      KCOM         <-- abhängig vom Transaktionsstatus:
                KP        nur erlaubt bei KCTAST=O oder C
                RE        beendet die Transaktion bei KCTAST=P bzw. leitet Transaktionsende
                          ein bei KCTAST=O oder C
                FI        Ende des Auftragnehmer-Vorgangs, sonst wie PEND RE
                ER/FR     Ende des Auftragnehmer-Vorgangs, die Transaktion wird
                          zurückgesetzt, der Auftraggeber wird informiert
                PA/PR     keine Besonderheiten, kann nicht verwendet werden, um Nachricht an
                          den Auftraggeber zu senden
      KCRN         <-- ggf. (bei PEND KP oder PEND RE) Name des Folge-Teilprogramms des
                       Auftragnehmer-Vorgangs
```

3. Folge-Teilprogramm des Auftraggeber-Programms (zweiter Teil)

```
INIT
      Auswerten:
      KCRPI        --> Vorgangs-Identifikation des Auftragnehmer-Vorgangs
      KCRMF        --> Formatkennzeichen aus 1. MPUT des Auftragnehmers
MGET  Nachricht vom Auftragnehmer lesen
      Versorgen:
      KCOM         <-- NT
      KCLA         <-- Länge des Nachrichtenbereichs
      KCRN         <-- Vorgangs-Identifikation aus KCRPI des INIT-Aufrufs
      KCMF         <-- Formatkennzeichen aus KCRMF des INIT-Aufrufs
      Auswerten:
      KCRLM        --> aktuelle Länge der Eingabenachricht
      KCRMF        --> wenn noch Folgeteile vorliegen: das Formatkennzeichen des nächsten
                       Nachrichtenteils,
                       sonst: das Formatkennzeichen des gelesenen Teils.
      KCRDF        --> Wert vom zugehörigen MPUT des Auftragnehmer-Vorgangs
      KCRPI        --> Vorgangs-Iidentifikation, falls weitere Nachrichtenteile vorliegen
      KCRST 1.Byte --> Vorgangs-Status des Auftragnehmer-Vorgangs:
                       O   Auftragnehmer-Vorgang ist offen
                       C   Auftragnehmer-Vorgang ist beendet (PEND FI)
      KCRST 2.Byte --> Transaktionsstatus des Auftragnehmer-Vorgangs:
                       O   Transaktion ist offen (PEND KP)
                       P   Auftragnehmer hat Transaktionsende angefordert (mit PEND RE oder
                           FI, PTC-Zustand)
                       C   Transaktion ist abgeschlossen (PEND RE oder FI)
MPUT  an das Terminal
      Versorgen:
      KCRN         <-- mit Leerzeichen
      KCOM         <-- mit NT oder NE
      KCLM         <-- mit der Länge der Nachricht
      KCMF         <-- mit dem Formatkennzeichen oder Leerzeichen
      KCDF         <-- ggf. mit einer Bildschirmfunktion
PEND  Ende des Folge-Teilprogramms des Auftragnehmer-Vorgangs
      Versorgen:
      KCOM         <-- abhängig von Status-Anzeigen in KCRST
                FI       nur bei KCVGST=C erlaubt, beendet Vorgang und Transaktion
                RE       beendet die Transaktion bei KCTAST=P, nicht erlaubt bei KCTAST=O
                         (weil Nachricht an das Terminal)
                ER/FR    setzt Transaktion zurück, auch der Auftragnehmer-Vorgang wird
                         zurückgesetzt und beendet;
                         einzige Ausnahme: KCTAST=C und KCVGST=C.
                KP       nicht zu empfehlen, wenn KCTAST=P
                PA/PR    verboten, da Nachricht an das Terminal gesendet wurde.
      KCRN         <-- ggf. (bei PEND KP oder PEND RE) Name des Folge-Teilprogramms des
                       Auftraggeber-Vorgangs
```

5.4 Verteilte Dialoge über OSI TP

Für die verteilte Verarbeitung zwischen Anwendungen wurde von der ISO (International Organization for Standardization) das Protokoll OSI TP definiert (**O**pen **S**ystems **I**nterconnection **T**ransaction **P**rocessing).

OSI TP gehört zu der Schicht 7 des OSI-Referenzmodells und wurde unter der Bezeichnung ISO/IEC 10026 als internationaler Standard verabschiedet. OSI TP erlaubt es insbesondere verteilte, d.h. rechnerübergreifende, Transaktionen kontrolliert abzuwickeln. Darüber hinaus kann dieses Protokoll jedoch auch eingesetzt werden, wenn zwischen zwei Anwendungen lediglich Daten ausgetauscht werden, die keiner Transaktionssicherheit unterliegen. Solche Anwendungsfälle liegen häufig bei Client/Server-Kommunikation vor.

5.4.1 Funktionseinheiten

Die Funktionen von OSI TP sind in mehrere Funktionsgruppen, so genannte Functional Units (FU) untergliedert. Je nach den an die Kommunikation mit einer Partner-Anwendung gestellten Anforderungen können für die Kommunikation verschiedene Funktionen ausgewählt werden. openUTM unterstützt die folgenden Functional Units:

Dialogue

Die Functional Unit Dialogue wird bei jeder Kommunikation über das OSI TP-Protokoll benötigt. Sie bietet Funktionen zum Auf- und Abbau von Dialogen, sowie zum Senden von Fehlernachrichten.

Dialoge werden mit dem KDCS-Aufruf APRO aufgebaut. Beim APRO-Aufruf können die OSI TP-Funktionskombinationen ausgewählt werden, die für diesen Dialog genutzt werden sollen. Normal beendet werden Dialoge mit einem PEND FI-Aufruf. Abnormal beendet werden Dialoge mit dem Aufruf PEND ER oder CTRL AB. Ein MPUT EM löst das Protokollelement TP-U-ERROR aus, ein CTRL AB oder PEND ER das Protokollelement TP-ABORT.

Polarized Control

Die Functional Unit Polarized Control dient dazu, das Senderecht eines Dialogs zu verwalten. Jedem Dialog wird ein Senderecht zugeordnet, welches zu einer Zeit nur einer der Kommunikationspartner besitzen kann.

Bei UTM-Vorgängen wechselt das Senderecht für einen Dialog am Ende jedes Verarbeitungsschritts, in dem eine Nachricht an den Dialog-Partner gesendet wurde: openUTM erzeugt implizit das Protokollelement TP-GRANT-CONTROL.

Handshake

Mit den Handshake-Funktionen können die Kommunikationspartner die Verarbeitung innerhalb eines Dialogs auf Anwendungsebene koordinieren. Diese Funktion erlaubt es, Verarbeitungsquittungen anzufordern und positive bzw. negative Quittungen zu senden. Eine anwendungsübergreifende Transaktionssicherung ist mit dieser Funktion nicht verbunden.

Eine Handshake-Anforderung kann mit dem Aufruf MPUT HM erzeugt werden. Eine von der Partner-Anwendung gesendete Handshake-Anforderung wird nach einem MGET-Aufruf angezeigt. Da bei Verwendung der KDCS-Schnittstelle Nachrichten erst gesendet werden, wenn das Senderecht abgegeben wird, wird von openUTM nur das OSI TP-Protokollelement TP-HANDSHAKE-AND-GRANT-CONTROL erzeugt, nicht jedoch TP-HANDSHAKE.

Eine positive Quittung auf eine Handshake-Anforderung sendet openUTM implizit vor der nächsten Nachricht an den Partner, von dem die Anforderung stammt, spätestens jedoch beim nächsten Transaktionsende.

Eine negative Quittung auf eine Handshake-Anforderung wird mit einem MPUT EM-Aufruf gesendet.

Das Resultat der Handshake-Anforderung kann der anfordernde Vorgang mit einem MGET-Aufruf lesen.

Commit und Chained Transactions

Die Commit Functional Unit stellt die Funktionen zur Verfügung, die zum Bilden von verteilten Transaktionen erforderlich sind. Dazu gehört insbesondere das Vor- und Zurücksetzen von verteilten Transaktionen. Zusammen mit dieser Funktion muss immer auch die Chained Transactions Functional Unit ausgewählt werden. Falls mit globaler Transaktionssicherung gearbeitet werden soll, werden für diesen Dialog ausschließlich verteilte Transaktionen abgewickelt.

Für diese Funktionsgruppen sind die Aufrufe MPUT, CTRL und PEND/PGWT von Bedeutung.

Die Operationsmodifikation des PEND/PGWT-Aufrufs entscheidet in Kombination mit dem Ziel der in dem letzten Verarbeitungsschritt erzeugten MPUT-Nachrichten darüber, ob ein TP-PREPARE gesendet wird und falls ja, ob dies mit DATA-PERMITTED=TRUE oder FALSE geschieht. Ein TP-PREPARE kann jedoch auch durch den Aufruf CTRL PR erzeugt werden.

Auf die gleiche Weise werden die OSI TP-Protokollelemente TP-DEFER(GRANT-CONTROL) und TP-DEFER(END-DIALOGUE) ausgelöst. Letzteres Protokollelement kann auch gezielt durch den Aufruf CTRL PE erzeugt werden.

Ein Transaktionsende wird durch einen PEND-Aufruf mit entsprechender Operationsmodifikation oder den Aufruf PGWT CM angefordert. Das Protokoll zur Abwicklung desTwo-Phase-Commit wird von openUTM ohne Beteiligung des Anwendungsteilprogramms abgehandelt.
Eine verteilte Transaktion kann durch PEND RS oder PGWT RB zurückgesetzt werden. PGWT RB **muss** verwendet werden, wenn die vorherige Transaktion mit PGWT CM beendet wurde.

Die Protokolle zum Rücksetzen der verteilten Transaktion wickelt openUTM ohne Beteiligung des Anwendungsteilprogramms ab.

Heuristische Entscheidungen von Kommunikationspartnern werden im Transaktionsstatus nach einem MGET-Aufruf angezeigt.

Recovery

Die Recovery Functional Unit stellt die Dienste zur Verfügung, die bei verteilter Transaktionsverarbeitung nach einem Kommunikationsausfall zur Resynchronisation der unterbrochenen Transaktion benötigt werden. Damit kann auch in einem solchen Fall die globale Datenkonsistenz gewährleistet werden. Ein Fortsetzen einer unterbrochenen Verbindung (Dialog-Wiederanlauf) wird jedoch von OSI TP nicht ermöglicht.

Die Dienste der Recovery Funktionseinheit werden von openUTM intern genutzt. Sie sind für ein Anwendungsprogramm nicht direkt zugänglich.

5.4.2 Programmierhilfen

Ein Teilprogramm erhält über verschiedene Anzeigen Informationen über seine Kommunikationspartner. Diese Informationen erlauben es ihm, gezielt auf bestimmte Situationen zu reagieren. Die Informationen werden von openUTM nach den Aufrufen INIT und MGET zur Verfügung gestellt.

Ein Auftragnehmer erhält nach dem INIT-Aufruf angezeigt, dass er von einem OSI TP-Partner aufgerufen wurde und welche OSI TP-Funktionen der Auftraggeber für den Dialog ausgewählt hat. Ebenfalls nach dem INIT-Aufruf erfährt ein Auftragnehmer in weiteren Anzeigen, ob ihn sein Auftraggeber aufgefordert hat, Transaktions- oder Dialogende anzufordern und ob er dem Auftraggeber in der laufenden Transaktion noch eine Nachricht schicken muss.

Nach einem INIT- oder MGET-Aufruf erfährt ein Teilprogrammlauf von welchem Kommunikationspartner eine Nachricht gelesen werden kann und mit welcher abstrakten Syntax diese Nachricht gesendet wurde. Beim Empfang der Nachrichten muss sich das Teilprogramm an die von openUTM vorgegebene Reihenfolge der Nachrichten halten.
Bei einem MGET-Aufruf erfährt das Teilprogramm den Typ der empfangenen Nachricht und es erhält Informationen über den Vorgangs- und Transaktionsstatus des Kommunikationspartners.

Die im Einzelnen nach einem INIT oder MGET verfügbaren Informationen werden im Folgenden genauer vorgestellt.

Anzeige der für den Dialog ausgewählten OSI TP-Funktionen

Beim INIT-Aufruf werden in die Felder KCCP und KCOF1 im KBKOPF wichtige Informationen eingetragen.

In KCCP steht, welches Kommunikationsprotokoll der Partner verwendet: bei OSI TP ist hier ’2’ eingetragen. Daran kann das Teilprogramm erkennen, dass es von einem OSI TP-Auftraggeber aufgerufen wurde.

In KCOF1 stehen Informationen über die OSI TP-Funktionen, die für den Dialog mit dem Auftraggeber zur Verfügung stehen. Die Werte im Feld KCOF1 haben folgende Bedeutung.

B Basisfunktionen

Für den Dialog mit dem Auftraggeber sind die Functional Units Dialogue und Polarized Control ausgewählt.

H Basis- und Handshake-Funktionen

Für den Dialog mit dem Auftraggeber sind die Functional Units Dialogue, Polarized Control und Handshake ausgewählt.

C Basis- und Commit-Funktionen mit Chained Transactions

Für den Dialog mit dem Auftraggeber sind die Functional Units Dialogue, Polarized Control, Commit und Chained Transactions ausgewählt.

O (other combination)

Für den Dialog mit dem Auftraggeber wurde keine Standardkombination ausgewählt. Wurde INIT PU aufgerufen und OSI TP-Informationen angefordert, dann werden die verfügbaren OSI TP-Funktionen im Nachrichtenbereich angezeigt.

Anforderung von Transaktions- oder Vorgangsende durch Auftraggeber

Nach einem INIT PU-Aufruf wird in einem Auftragnehmer-Vorgang im Feld KCENDTA im Nachrichtenbereich angezeigt, ob der lokale Vorgang von seinem Auftraggeber aufgefordert wurde, die Transaktion oder den Vorgang zu beenden und welche Ausprägung des PEND-Aufrufs hierfür zu verwenden ist. Eine Aufforderung zum Transaktions- oder Vorgangsende muss der lokale Vorgang spätestens zum Ende des Verarbeitungsschritts befolgen, in dem er das nächste Mal eine Nachricht an den Auftraggeber sendet.

Folgende Werte sind möglich:

'BLANK'	keine Vorschrift bezüglich der Beendigung des Verarbeitungsschritts.
O	am Ende des Verarbeitungsschritts darf kein Transaktionsende anfordert werden.
R	die Transaktion und der Dialog-Schritt müssen abgeschlossen werden, der Vorgang darf nicht beendet werden (PEND RE oder PGWT CM mit vorherigem MPUT an den Auftraggeber). Das Senderecht für den Dialog zum Auftraggeber liegt am Transaktionsende beim Auftraggeber. Für alle anderen Dialoge besitzt der lokale Vorgang das Senderecht zum Transaktionsende.
S	die Transaktion muss abgeschlossen werden, der Dialog-Schritt darf nicht abgeschlossen werden (PEND SP oder PGWT CM ohne vorherigen MPUT an den Auftraggeber). Das Senderecht für den Dialog zum Auftraggeber liegt am Transaktionsende beim lokalen Vorgang.
C	die Transaktion muss abgeschlossen werden, der Vorgang darf nicht beendet werden (PEND RE/SP oder PGWT CM). Das Senderecht auf dem Dialog zum Auftraggeber liegt am Transaktionsende beim lokalen Vorgang. Auf einem anderen Dialog kann zum Transaktionsende das Senderecht an einen Auftragnehmer abgegeben werden; dies geschieht mit MPUT an einen Auftragnehmer und anschließendem PEND RE oder PGWT CM.
F	die Transaktion und der Vorgang müssen abgeschlossen werden (PEND FI).

Anzeige des Senderechts für den Dialog zum Auftraggeber

Nach einem INIT PU-Aufruf wird in einem Auftragnehmer-Vorgang im Feld KCSEND im Nachrichtenbereich angezeigt, ob der lokale Vorgang in dem Verarbeitungsschritt eine Nachricht an den Auftraggeber senden darf. Folgende Werte sind möglich.

Y	Das Senden einer Nachricht an den Auftraggeber ist am Ende des Dialog-Schritts notwendig. Das Senden einer Nachricht an den Auftraggeber ist am Ende der Transaktion in diesem Fall auch notwendig, wenn KCENDTA "S" enthält. Diese Kombination (KCENDTA=S und KCSEND=Y) kann nur bei heterogener Kopplung auftreten.
N	An den Auftraggeber darf keine Nachricht gesendet werden. Es können jedoch Nachrichten an Auftragnehmer gesendet werden, in diesem Fall muss jedoch die Transaktion über das Ende des Verarbeitungsschritts hinaus offen bleiben.

Anzeige des Typs einer empfangenen Nachricht

Nach einem MGET-Aufruf wird im Feld KCRMGT des Rückgabebereichs der Typ der Nachricht angezeigt. Folgende Werte können auftreten:

C (confirm)

Eine positive Handshake-Quittung wurde empfangen.

E (error)

Eine Fehlermeldung oder eine negative Handshake-Quittung wurde empfangen.

H (handshake)

Eine Handshake-Anforderung wurde empfangen.

M (message)

Es wurde eine normale Benutzernachricht empfangen.

Vorgangs-Status

Nach einem MGET-Aufruf wird einem Teilprogramm im Feld KCVGST/kcpcv_state des Rückgabebereichs der Vorgangs-Status des Kommunikationspartners angezeigt, von dem eine Nachricht empfangen wurde. Aus dieser Anzeige kann der lokale Vorgang Rückschlüsse auf den Dialog mit diesem Partner ziehen.

Folgende Werte sind möglich:

C (closed)

Der Auftragnehmer hat den Dialog beendet.

D (disconnected)

die Kommunikation mit dem Auftragnehmer wurde wegen eines Verbindungsverlusts beendet.

I (inactive)

Der Auftragnehmer-Vorgang konnte nicht gestartet werden, weil z.B. der TAC unbekannt ist.

O (open)

Der Partner-Vorgang ist offen, d.h. es wurde noch kein Dialogende angefordert.

P (pending end dialogue)

Dieser Status kann nur bei heterogener Kopplung und bei Dialogen auftreten, für die die Commit Funktionalität nicht ausgewählt ist.

Der Auftragnehmer möchte die Kommunikation beenden. Wenn der Auftraggeber nicht einverstanden ist, kann er die Kommunikation mit einem MPUT EM fortsetzen.

T (time out)

Es konnte keine Verbindung in der generierten Wartezeit belegt werden oder der Auftragnehmer hat in der generierten Wartezeit keine Nachricht geschickt; der Dialog mit dem Auftragnehmer wird beendet.

Z (error)

Der Dialog mit dem Auftragnehmer wurde wegen eines Fehlers beendet.

Bei den Stati D, I und T wird keine Nachricht übergeben. Im Auftragnehmer-Vorgang ist der Vorgangs-Status immer O.

Transaktionsstatus

Nach einem MGET-Aufruf wird einem Teilprogramm im Feld KCTAST/kcpta_state des Rückgabebereichs der Transaktionsstatus des Kommunikationspartners angezeigt, von dem eine Nachricht empfangen wurde. Aus dieser Anzeige kann der lokale Vorgang Rückschlüsse auf den Dialog mit diesem Partner ziehen.
Folgende Werte sind möglich:

H (heuristic hazard)

Weil die Kommunikation mit mindestens einem Kommunikationspartner unterbrochen wurde, besteht Unsicherheit über den Ausgang der Transaktion. Es kann nicht ausgeschlossen werden, dass einer der an der letzten Transaktion beteiligten Kommunikationspartner eine heuristische Entscheidung getroffen hat, die im Widerspruch zum tatsächlichen Ausgang der Transaktion steht.

I (inactive)

Die Transaktion beim Auftragnehmer ist inaktiv, z.B. weil der TAC ungültig ist oder weil keine Verbindung in der generierten Wartezeit belegt werden konnte.

M (mismatch)

Die Transaktion im entfernten Vorgang konnte nicht mit der Transaktion im lokalen Vorgang synchronisiert werden. Diese Situation kann nach einem Timeout auftreten.

Ein Mismatch entsteht auch dann, wenn mindestens einer der an der Transaktion beteiligten Kommunikationspartner eine heuristische Entscheidung getroffen hat, die im Widerspruch zum tatsächlichen Ausgang der Transaktion steht.

O (open)

die Transaktion im entfernten Vorgang ist offen.

P (prepare to commit)

Der ferne Vorgang hat Transaktionsende eingeleitet oder er fordert den lokalen Vorgang auf, das Transaktionsende einzuleiten.

R (reset)

Die Transaktion im entfernten Vorgang wurde zurückgesetzt.

U (unknown)

Der Transaktionsstatus ist unbekannt. Der Wert ist nur möglich bei Dialogen, für die die Commit Funktionalität nicht ausgewählt wurde.

Im Auftragnehmer-Vorgang sind nur folgende Transaktionsstati möglich:

mit Functional Unit Commit: O, P
ohne Funcional Unit Commit: U

5.4.3 Programmierregeln für Dialoge ohne Functional Unit Commit

Transaktionsende

Bei einer Kommunikation gemäß dem Cooperative Processing können die Kommunikationspartner unabhängig voneinander ein Transaktionsende anfordern. Bei dieser Art der Verarbeitung gibt es ausschließlich lokale Transaktionen, d.h. das Transaktionsende in einem Vorgang hat keine Auswirkungen auf die Transaktion im Partner-Vorgang.

Vorgangsende

Ein Auftragnehmer-Vorgang darf zu jeder Zeit seinen Vorgang beenden.

Ein Auftraggeber-Vorgang darf erst dann beendet werden, nachdem alle Dialoge mit seinen Auftragnehmer-Vorgängen beendet sind.

PEND FC (Vorgangs-Kettung) in Auftragnehmer-Vorgängen ist verboten.

5.4.4 Programmierregeln für Dialoge mit Functional Unit Commit

Bevor die Regeln aufgeführt werden, die ein Vorgang zu beachten hat, der an einer verteilten Transaktion beteiligt ist, sollen zunächst einige Begriffe erläutert werden.

Begriffsklärungen

Datentransferphase

Ein Vorgang befindet sich in der Datentransferphase, solange er weder zum Transaktionsende aufgefordert wurde noch seinerseits seine Auftragnehmer zum Transaktionsende aufgefordert hat.

Senderecht

Während der Datentransferphase existiert auf jedem Dialog ein Senderecht, das zu einer Zeit genau einem der beiden Kommunikationspartner zugeordnet ist.
Der Vorgang, der das Senderecht auf einem Dialog besitzt, darf mit MPUT eine Nachricht an den Partner-Vorgang senden. Mit dem Senden einer Nachricht wechselt das Senderecht zu dem Kommunikationspartner, es sei denn, er hat dies explizit mit einem CTRL-Aufruf unterbunden (CTRL PR oder PE und KCNORPLY=Y).

Ein Vorgang kann in einem Verarbeitungsschritt entweder das Senderecht für den Dialog mit dem Auftraggeber abgeben, oder aber ein oder mehrere Senderechte für Dialoge zu Auftragnehmern.

Senderecht zum Transaktionsende

Das Senderecht zum Transaktionsende regelt, welcher der beiden Kommunikationspartner nach Abschluss der laufenden Transaktion das Senderecht besitzt.
Normalerweise liegt das Senderecht zum Transaktionsende bei dem Auftraggeber-Vorgang. Dieser kann dieses Senderecht jedoch mit einer MPUT-Nachricht und anschließendem PEND RE an den Auftragnehmer-Vorgang abgeben.
Dabei ist zu beachten, dass ein Vorgang zum Transaktionsende das Senderecht auf höchstens einem Dialog abgeben darf.

Transaktionsende

Ein Vorgang darf Transaktionsende anfordern, wenn ihn sein Auftraggeber-Vorgang dazu aufgefordert hat und er das Senderecht zum Transaktionsende auf höchstens einem Dialog nicht besitzt.

Vorgangsende

Ein Vorgang darf Vorgangsende anfordern, wenn er von seinem Auftraggeber-Vorgang dazu aufgefordert wurde und er im laufenden Verarbeitungsschritt keine Nachricht an einen Auftragnehmer gesendet hat.

Weitere Programmierregeln

- In der Datentransferphase besitzt der lokale Vorgang während eines Teilprogrammlaufs das Senderecht auf allen Dialogen.
- Ein Vorgang kann in einem Verarbeitungsschritt nicht gleichzeitig Nachrichten an seinen Auftraggeber und an Auftragnehmer senden.
- Bleibt die Transaktion zum Ende des Verarbeitungsschritts offen, dann kann der lokale Vorgang in diesem Verarbeitungsschritt an mehrere Auftragnehmer gleichzeitig Nachrichten senden.

- Ein Vorgang darf das Senderecht zum Transaktionsende an höchstens einen Partner abgeben. Folgerungen aus dieser Regel sind:
  - In einem Verarbeitungsschritt, der mit einer Transaktionsendeanforderung abgeschlossen wird, darf an höchstens einen Partner eine Nachricht gesendet werden.
  - Ein Zwischenknoten darf auf einem Dialog zu einem Auftragnehmer nur dann das Senderecht zum Transaktionsende abgeben, wenn er auf dem Dialog mit dem Auftraggeber das Senderecht zum Transaktionsende besitzt.
- Ein Auftragnehmer darf erst dann Transaktionsende anfordern, wenn ihn sein Auftraggeber dazu aufgefordert hat.

Regeln zur Verwendung der verschiedenen PEND-Varianten

- Ein PEND KP ist möglich, wenn in dem Verarbeitungsschritt nur Nachrichten an Partner gesendet wurden, die noch kein Transaktionsende angefordert haben.
- Ein PEND RE ist möglich, wenn in dem Verarbeitungsschritt
  - an höchstens einen Partner eine Nachricht gesendet wurde und
  - an diesen Partner nicht gleichzeitig eine Aufforderung zu Transaktions- oder Dialogende gesendet wurde und
  - der lokale Vorgang bereits eine Aufforderung zum Transaktionsende erhalten hat, oder der lokale Vorgang selbst die Wurzel im Transaktionsbaum ist.
- Ein PEND SP ist möglich, wenn in dem Verarbeitungsschritt
  - der lokale Vorgang bereits eine Aufforderung zum Transaktionsende erhalten hat, oder der lokale Vorgang selbst die Wurzel im Transaktionsbaum ist und
  - der lokale Vorgang für den Dialog mit dem Auftraggeber das Senderecht zum Transaktionsende besitzt und
  - keine Nachricht an einen Auftragnehmer gesendet wurde und
  - keine Nachricht an den Client gesendet wurde.
- Ein PEND FI ist möglich, wenn in dem Verarbeitungsschritt
  - der lokale Vorgang bereits eine Aufforderung zum Dialogende erhalten hat, oder der lokale Vorgang selbst die Wurzel im Transaktionsbaum ist und
  - kein Auftragnehmer existiert, der (mittels CTRL PR) zum Transaktionsende, aber nicht zum Vorgangsende aufgefordert wurde.
  - keine Nachricht an einen Auftragnehmer gesendet wurde.

Regeln zur Verwendung von PGWT-Varianten

- Ein PGWT KP ist möglich, wenn ein PEND KP erlaubt ist.
- Ein PGWT CM ist möglich
  - bei Ausgabe einer Dialog-Nachricht immer dann, wenn ein PEND RE zulässig ist.
  - ohne Dialog-Nachricht immer dann, wenn ein PEND SP zulässig ist.
- PGWT RB muss verwendet werden, wenn eine Transaktion, bei der der letzte Sicherungspunkt mit PGWT CM gesetzt wurde, zurückgesetzt werden soll, ohne dass der OSI TP-Dialog beendet wird.

Programmierempfehlung

Bei der verteilten Transaktionsbearbeitung über das OSI TP-Protokoll wird eine verteilte Transaktion am günstigsten auf folgende Weise beendet.

Wenn der oberste Auftraggeber im Transaktionsbaum die verteilte Transaktion beenden möchte, richtet er an jeden seiner Auftragnehmer einen CTRL PR und einen MPUT-Aufruf und beendet anschließend den Verarbeitungsschritt mit PEND/PGWT KP. Nachdem die Antworten von seinen Auftragnehmern eingetroffen sind, schließt der oberste Auftraggeber die Transaktion ab.

Erhält ein Zwischenknoten im Transaktionsbaum die Aufforderung zum Transaktionsende, dann richtet er seinerseits an jeden seiner Auftragnehmer einen CTRL PR und einen MPUT-Aufruf und beendet anschließend den Verarbeitungsschritt mit PEND/PGWT KP. Nachdem die Antworten von seinen Auftragnehmern eingetroffen sind, sendet der Zwischenknoten eine Antwort an seinen Auftraggeber und beendet den Dialog-Schritt und die Transaktion.

Erhält ein unterster Auftragnehmer im Transaktionsbaum die Aufforderung zum Transaktionsende, dann sendet er eine Antwort an seinen Auftraggeber und beendet den Dialog-Schritt und die Transaktion.

Wird diese Regel befolgt, dann beginnt die nächste Transaktion jeweils mit einem Teilprogrammlauf im obersten Auftraggeber-Vorgang.

5.4.5 Programmierregeln bei der Kommunikation mit BeanConnect

Wenn openUTM über BeanConnect einen OLTP Message-Driven Bean in einem Java EE Application Server aufruft, ist Folgendes zu beachten:

- Die Handshake-Funktionseinheit darf nicht ausgewählt werden.
- Es sind nur Einschritt-Dialoge erlaubt, d.h. bei Dialogen ohne Functional Unit Commit beendet der Auftragnehmer nach dem Senden der Antwort den Dialog.
- Bei Dialogen mit Functional Unit Commit muss der Auftraggeber den Auftragnehmer entweder sofort mit dem Senden der Nachricht zum Dialogende auffordern oder nach Empfang der Antwort. Fordert der Auftraggeber erst nach Empfang der Antwort zum Dialogende auf, hat der Auftragnehmer im Fehlerfall noch die Möglichkeit, eine (Fehler-)Nachricht an den Auftraggeber zu senden und der Auftraggeber kann die Transaktion gegebenenfalls selbst zurücksetzen.

5.4.6 Besonderheiten bei Rollback und Wiederanlauf

Das OSI TP-Protokoll ermöglicht es, wahlweise mit globaler Transaktionssicherung zu arbeiten (Functional Unit Commit) oder ohne globale Transaktionssicherung (Cooperative Processing).

OSI TP mit Functional Unit Commit

Falls bei OSI TP die Functional Unit Commit gewählt ist, gibt es beim Rücksetzen von Transaktionen (Rollback) mit PEND RS keinen Unterschied zu LU6.1.

Beim Vorgangs-Wiederanlauf nach PEND RS gleicht das Verhalten dem bei LU6.1- mit einer Ausnahme: Das OSI TP-Protokoll ermöglicht keinen Wiederanlauf eines unterbrochenen Dialogs.

Zusätzlich kann bei OSI TP mit PGWT CM ein Sicherungspunkt gesetzt werden. Die nachfolgende Transaktion darf dann nur mit PGWT RB zurücksetzt werden. In diesem Fall wird immer in das Teilprogramm zurückgekehrt, das den PGWT RB abgesetzt hat. Es findet kein Vorgangs-Wiederanlauf statt.

OSI TP ohne Functional Unit Commit

Falls die Functional Unit Commit nicht ausgewählt ist, wird der Auftraggeber-Vorgang bei einem Fehler im Auftragnehmer-Vorgang oder bei Verbindungsverlust **nicht** automatisch zurückgesetzt: Er wird mit dem beim letzten PEND angegebenen Teilprogramm fortgesetzt. Er erhält jedoch eine Fehlernachricht (mit Vorgangs-Status "Z" und Transaktionsstatus "U"), falls er auf ein Ergebnis vom Auftragnehmer wartet, und kann ebenfalls mit Rücksetzen reagieren.

Bei einem Fehler im Auftraggeber-Vorgang setzt openUTM die Transaktionen in Auftraggeber- und Auftragnehmer-Vorgang zurück und beendet den Auftragnehmer-Vorgang. Hat der Auftraggeber-Vorgang bereits einen Sicherungspunkt erreicht, wird nach dem Rücksetzen der Transaktion ein Vorgangs-Wiederanlauf durchgeführt. Das Folge-Teilprogramm erhält eine Fehlernachricht mit Vorgangs-Status "Z" und Transaktionsstatus "U".

Ein globaler Vorgangs-Wiederanlauf ist nicht möglich, da keine gemeinsamen Sicherungspunkte existieren.

Bei den Aufrufen zum programmierten Rücksetzen ist Folgendes zu beachten:

- PEND RS

  Beim Aufruf im Auftraggeber-Vorgang:

  - Alle Auftragnehmer-Vorgänge, mit denen ohne Functional Unit Commit kommuniziert wird, werden beendet.

  Bei einem Aufruf in einem Auftragnehmer-Vorgang:

  - Falls die vorhergehende Transaktion mit PEND SP beendet wurde, wird bei einem PEND RS die lokale Transaktion zurückgesetzt und der Vorgang mit dem beim PEND SP spezifizierten Folge-Teilprogramm fortgesetzt.
  - Falls die vorhergehende Transaktion **nicht** mit PEND SP beendet wurde und der Vorgang unter einer Benutzerkennung ohne Wiederanlaufeigenschaft läuft, so wird der Vorgang auf den letzten Sicherungspunkt zurückgesetzt und der Dialog mit dem Auftraggeber beendet.
  - In allen anderen Fällen beendet openUTM den Vorgang mit PEND FR.
- PGWT RB

  PGWT RB setzt die aktuelle Transaktion zurück und setzt das Teilprogramm fort.

- PEND ER/FR

  Beim Aufruf im Auftraggeber-Vorgang gibt es keine Besonderheit: Die Transaktionen in Auftraggeber und allen seinen Auftragnehmern werden zurückgesetzt und die Auftragnehmer beendet.

  Beim Aufruf im Auftragnehmer -Vorgang wird der Auftragnehmer zurückgesetzt und beendet, der Auftraggeber-Vorgang jedoch mit dem beim letzten PEND angegebenen Teilprogramm fortgesetzt. Der Auftraggeber-Vorgang kann die vom Auftragnehmer mit MPUT bereitgestellte Nachricht lesen.

> ¡ Um auch beim Rücksetzen einer Transaktion einen konsistenten Ablauf zu gewährleisten, wird empfohlen, innerhalb einer verteilten Transaktion entweder nur PGWT- oder nur PEND-Aufrufe zu verwenden.

- RSET

  Der RSET-Aufruf wirkt immer nur auf den lokalen Vorgang. Die Einstellung RSET=GLOBAL der KDCDEF-Anweisung UTMD hat keine Wirkung. Diese Einstellung wirkt nur bei verteilter Verarbeitung mit globaler Transaktionssicherung (siehe "Programmiertes Rücksetzen").

5.4.7 Nutzung bestehender Teilprogramme für OSI TP-Kommunikation

Bestehende UTM-Teilprogramme, die nicht speziell für die Kommunikation über OSI TP konzipiert sind, lassen sich unter bestimmten Randbedingungen (siehe unten) unverändert für die Kommunikation über OSI TP nutzen. Ein und derselbe Service kann so beispielsweise sowohl von Clients als auch von anderen Vorgängen genutzt werden. Dadurch ermöglicht openUTM hohe Flexibilität bei der Verteilung von Anwendungen.

Bei der Verwendung bestehender Teilprogramme für OSI TP-Kommunikation lassen sich die in den folgenden Abschnitten dargestellten Fälle unterscheiden.

Teilprogramme für die Kommunikation mit Clients als OSI TP-Auftragnehmer

Teilprogramme, die für die Kommunikation mit Clients konzipiert sind, lassen sich unverändert als Auftragnehmer für die Kommunikation mit OSI TP-Partnern verwenden. Folgende Punkte sind dabei zu beachten:

- Unterschiedliche Rückgaben im KB-Kopf

  Der Kommunikationspartner des Auftragnehmer-Vorgangs ist der Auftraggeber-Vorgang, nicht der Benutzer am Terminal. Daher bekommt der Auftragnehmer-Vorgang im KB-Kopf nicht den Namen des LTERM-Partners sondern den Namen des OSI-LPAP-Partners. Die Versorgung des Feldes KCBENID/kcuserid ist abhängig vom verwendeten Security-Typ. Der Security-Typ wird im Auftraggeber beim APRO-Aufruf ausgewählt (im Feld KCSECTYP):

  - Bei Security-Typ "N" (None) wird keine Benutzerkennung an den Auftragnehmer übergeben. KCBENID /kcuserid enthält an Stelle einer Benutzerkennung den Namen der Association.
  - Bei Security-Typ "P" (Program) enthält KCBENID/kcuserid die Benutzerkennung, die im Auftraggeber beim APRO-Aufruf angegeben wurde.
  - Bei Security-Typ "S" (Same) enthält KCBENID/kcuserid die Benutzerkennung, unter der der Auftraggeber gestartet wurde.
- Unterschiedliche Zuordnung von TLS und ULS

  Schreib- und Leseaufrufe für TLS beziehen sich im Auftragnehmer-Vorgang auf den TLS des OSI-LPAP-Partners und nicht auf einen TLS eines LTERM-Partners; desgleichen beziehen sich Aufrufe für einen ULS nur bei Security-Typ P/S auf den ULS der Benutzerkennung, bei Security-Typ N dagegen auf den ULS der Association.
- Funktionstasten bei verteilter Verarbeitung nicht nutzbar

  Der Auftraggeber-Vorgang kann dem Auftragnehmer-Vorgang keine den Funktionstasten entsprechende Nachricht senden. Der Auftragnehmer-Vorgang kann deshalb beim MGET-Aufruf nie den entsprechenden KDCS-Returncode (19Z bis 39Z) erhalten.
- Ausweisleser im Auftragnehmer-Vorgang nicht nutzbar

  In einem Auftragnehmer-Vorgang enthält das Feld KCAUSWEIS/kccard immer Leerzeichen.
- abstrakte Syntax bei verteilter Verarbeitung über OSI TP

  Bei verteilter Verarbeitung über OSI TP wird das Formatkennzeichen zur Übergabe des Namens der abstrakten Syntax verwendet.

  Auftragnehmer-Programme, die für die Kommunikation mit Clients konzipiert sind, können nur dann ohne Anpassung für die verteilte Verarbeitung über OSI TP verwendet werden, wenn als abstrakte Syntax ausschließlich UDT Octet String Mapping verwendet wird. Für alle anderen abstrakten Syntaxen wären Anpassungen für Encodierung bzw. Decodierung der Nachrichten notwendig. Beim Nachrichtenaustausch muss das Feld KCMF/kcfn also immer Leerzeichen enthalten.

LU6.1-Auftragnehmer als OSI TP-Auftragnehmer

Auftragnehmer-Teilprogramme, die für die Kommunikation über LU6.1 geschrieben wurden, können nur unverändert als OSI TP-Auftragnehmer verwendet werden, wenn die Commit Functional Unit **nicht** ausgewählt wurde, und folgende Bedingungen erfüllt sind:

- Bei der Kommunikation über OSI TP wird als abstrakte Syntax ausschließlich UDT Octet String Mapping verwendet. Beim Nachrichtenaustausch muss das Feld KCMF/kcfn also immer Leerzeichen enthalten.
- In den Auftragnehmer-Programmen werden die Transaktions- und Vorgangs-Stati nicht ausgewertet.

Wird die Commit Functional Unit ausgewählt, so muss der Auftraggeber sowohl Transaktions- wie auch Vorgangsende als Erster anfordern und immer dann anfordern, wenn es der Auftragnehmer erwartet.

LU6.1-Auftraggeber als OSI TP-Auftraggeber

Auftraggeber-Teilprogramme, die für die Kommunikation über LU6.1 geschrieben sind, lassen sich nicht unverändert als OSI TP-Auftraggeber verwenden. Zumindest der APRO-Aufruf muss angepasst werden (Auswahl der OSI-Funktionen im Feld KCOF und ggf. im 2. Parameterbereich). Für verteilte Verarbeitung ohne globale Transaktionssicherung (Cooperative Processing) sind keine weiteren Anpassungen notwendig.

Bei verteilter Verarbeitung mit globaler Transaktionssicherung - d.h. wenn die Functional Unit "Commit" gewählt wurde - müssen die Programme auf jeden Fall hinsichtlich der Anforderung des Dialogendes erweitert werden (z.B. durch Einfügen von CTRL PE).

5.4.8 Besonderheiten bei heterogener Kopplung

Wenn Sie über OSI TP Ihre UTM-Anwendungen mit Transaktions-Anwendungen anderer Hersteller koppeln wollen, müssen Sie folgende Punkte beachten:

- Benutzerdaten beim Associationaufbau:
  Beim Verbindungsaufbau werden nur die für OSI TP und CCR benötigten Benutzerdaten ausgetauscht. Andere Benutzerdaten werden nicht gesendet und beim Empfang ignoriert.
- Benutzersyntaxen und CCR beim Associationaufbau:
  openUTM erlaubt nicht, dass bei openUTM generierte Syntaxen vom Partner abgelehnt werden oder beim Associationaufbau-Wunsch nicht angeboten werden. openUTM lehnt in so einem Fall den Associationaufbau ab.
- Verbindungsabbau mit A-ABORT:
  openUTM baut Verbindungen mit A-ABORT ab, nicht mit A-RELEASE.
- Channels:
  "Two-way-recovery" Channels werden von openUTM nicht unterstützt.
- Benutzerdaten beim TP-BEGIN-DIALOGUE-RI:
  Beim TP-BEGIN-DIALOGUE-RI können nur die für UTMSEC benötigten Benutzerdaten ausgetauscht werden. Anwenderprogramme haben keinen direkten Zugriff auf die Benutzerdaten. Andere Benutzerdaten werden nicht gesendet und beim Empfang ignoriert.
- Keine Benutzerdaten beim TP-BEGIN-DIALOGUE-RC, TP-ABORT-RI:openUTM sendet keine Benutzerdaten mit den Protokollelementen. Empfangene Benutzerdaten gehen verloren.
- Keine Shared Control Funktionseinheit:
  Die Shared Control Funktionseinheit wird von openUTM nicht unterstützt, d.h. openUTM unterstützt die Profile ATP12, ATP22, ATP32 nicht.
- Unchained Transactions Funktionseinheit:
  openUTM als Auftraggeber verwendet die Funktionseinheit nicht. Umgekehrt ist es aber möglich, die Funktionseinheit in der Partner-Anwendung auszuwählen, falls die Partner-Anwendung als Auftraggeber fungiert. Die Partner-Anwendung muss aber die verteilte Transaktion vor der ersten Senderechtabgabe in dem Dialog beginnen und der Dialog muss mit der 1. Transaktion enden. Andernfalls beendet openUTM den Dialog abnormal.
- Recipient TPSU-Title:
  Ein Recipient TPSU-Title ist immer erforderlich beim TP-BEGIN-DIALOGUE-RI. Er darf maximal acht Zeichen lang sein und nicht vom Typ Integer, wenn openUTM der Empfänger ist.
- REQUEST-CONTROL-RI, HANDSHAKE-RI:
  Die Protokollelemente werden von openUTM nicht gesendet. Der Empfang von TP-HANDSHAKE-RI für einen Dialog-Vorgang bewirkt die abnormale Beendigung des Dialog-Vorgangs.
- maximale Benutzerdatenlänge: 32767 octets:
  openUTM sendet maximal 32767 Octets Benutzerdaten in einem Protokollelement. Wenn Benutzerdaten empfangen werden, die mehr als 32767 Octets enthalten, baut openUTM die Verbindung ab.
- Dialog-Beendigung ohne Commit:
  An einen Dialog-Auftragnehmer-Vorgang sollte kein TP-END-DIALOGUE-RI gesendet werden (Dialog-Beendigung von "oben"), da openUTM dann den Vorgang abnormal beendet. openUTM verwendet "Confirmed End Dialogue" nur bei der Übertragung von Asynchron-Nachrichten.

5.4.9 Beispiele: verteilte Dialoge über OSI TP

In diesem Abschnitt sollen anhand von Beispielen die verschiedenen Möglichkeiten vorgestellt werden, die sich an der Programmschnittstelle bieten, wenn mit der Commit Funktionalität und Chained Transactions gearbeitet wird.

Dabei wird zuerst der einfachste Fall betrachtet, in dem ein Auftraggeber mit einem Auftragnehmer kommuniziert. Als Zweites wird das Szenario auf die Kommunikation mit mehr als einem Auftragnehmer erweitert und abschließend werden die komplizierteren Fälle dargestellt, in denen ein Vorgang sowohl mit einem Auftraggeber als auch mit einem Auftragnehmer über das OSI TP-Protokoll kommuniziert.

Dabei werden die Abschnitte Datentransferphase, Transaktionsende und Dialogende getrennt betrachtet.

Am Ende dieses Abschnitts finden Sie noch Beispiele für die abnormale "Beendigung eines Auftragnehmers mit CTRL AB".

Erläuterungen zu den nachfolgenden Bildern

In den nachfolgenden Bildern wird der Kommunikationsablauf bei verteilten OSI TP-Dialogen schematisch dargestellt. Bei den Vorgängen sind jeweils nur die für die Kommunikation relevanten KDCS-Aufrufe angegeben, andere KDCS-Aufrufe sowie verarbeitende Anweisungen bleiben ausgespart.

Rechts neben den MGET-Aufrufen sind der Vorgangs- und Transaktions-Status wiedergegeben, die dem Teilprogramm nach dem MGET-Aufruf angezeigt werden. In den Beispielen werden hierfür die COBOL-Feldnamen verwendet, also KCVGST für den Vorgangs-Status und KCTAST für den Transaktionsstatus. Für C/C++ heißen die entsprechenden Felder *kcpcv_state* und *kcpta_state.*

Rechts neben den INIT-Aufrufen in den Auftragnehmer-Vorgängen sind die Felder KCENDTA und KCSEND angegeben, die ein Teilprogramm nach einem INIT PU-Aufruf auswerten kann.

Bei den MPUT-Aufrufen, die für einen Auftragnehmer bestimmt sind, und bei den CRTL-Aufrufen ist jeweils der Auftragnehmer-Vorgang bezeichnet, an den der Aufruf gerichtet ist; dies geschieht in der Form ">*x*", wobei *x* den Auftragnehmer repräsentiert. MPUT-Aufrufe ohne diese Angabe sind immer an den Auftraggeber oder den Client gerichtet.

Die Pfeile zwischen Auftraggebern und Auftragnehmern symbolisieren den Nachrichtenaustausch und den Protokollfluss.

Sicherungspunkte sind durch eine dicke durchgezogene Linie gekennzeichnet.

Wenn in den Beispielen in diesem Abschnitt PEND KP-Aufrufe angegeben sind, dann können diese jeweils durch einen PGWT KP-Aufruf ersetzt werden. Entsprechend kann statt PEND RE kann auch ein PGWT CM mit vorheriger MPUT-Nachricht und statt PEND SP ein PGWT CM ohne vorherige MPUT-Nachricht verwendet werden.

5.4.9.1 Ein Auftragnehmer

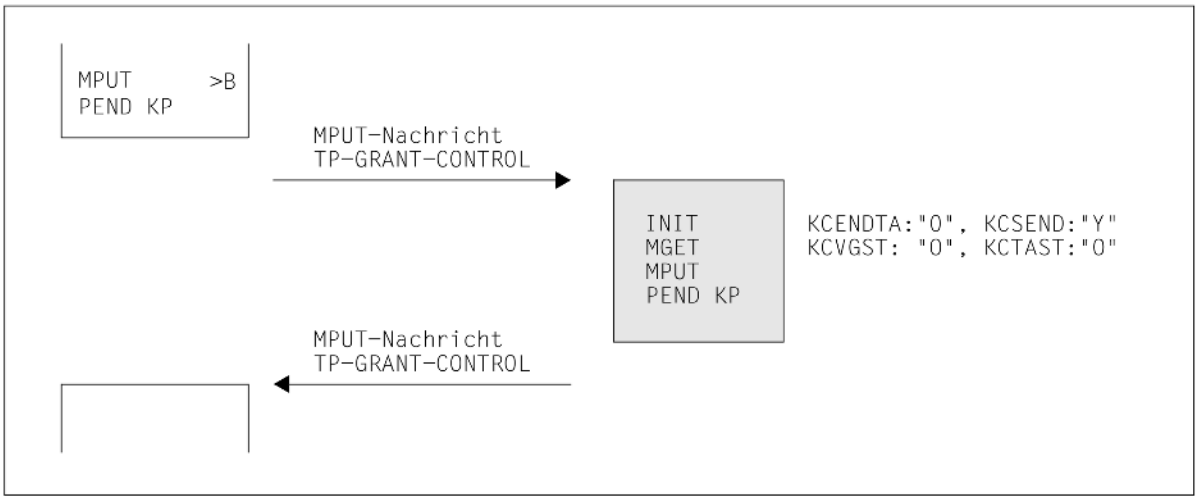
Der einfachste Fall ist gegeben, wenn ein Auftraggeber (A) genau einen Auftragnehmer (B) besitzt. In diesem Abschnitt sind alle möglichen Situationen für diesen Anwendungsfall dargestellt.

Datentransferphase

Die Phase des Nachrichtentransfers ist dadurch gekennzeichnet, dass keiner der Partner ein Transaktionsende anfordert. Dem Auftragnehmer bietet sich diese Möglichkeit ohnehin nur nach expliziter Anforderung des Auftraggebers.

Mit jeder Nachricht, die an den Partner gesendet wird, wechselt auch das Senderecht. Dem Auftragnehmer wird durch den Vorgangs-Status "O" und Transaktionsstatus "O" angezeigt, dass weder Transaktions- noch Vorgangsende angefordert sind.

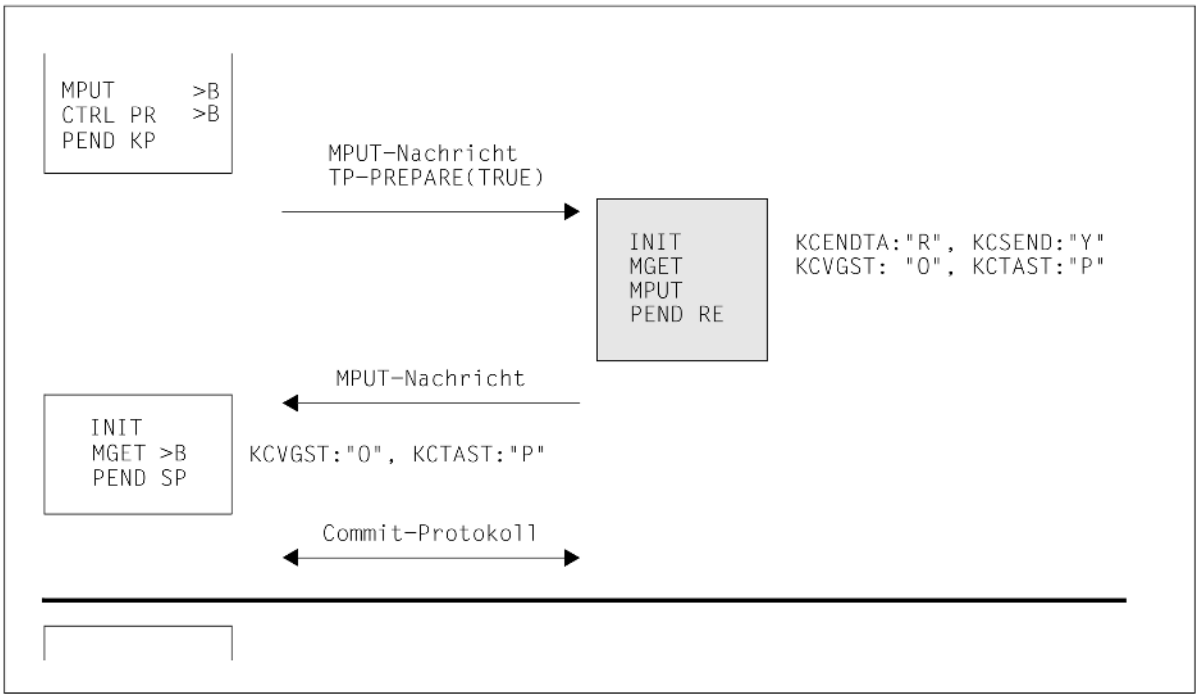
Beispiel 1: Nachricht an Auftragnehmer und PEND KP



Transaktionsende

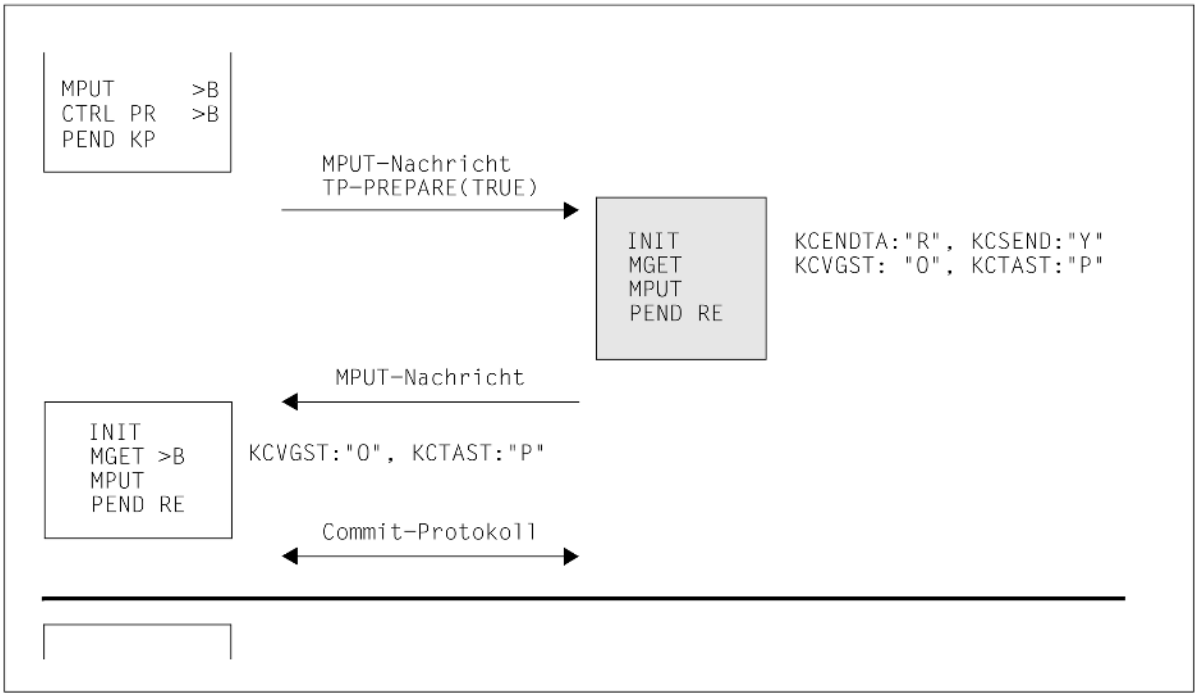
Ein Auftragnehmer darf nur dann einen Aufruf zum Transaktionsende absetzen, wenn er dazu von seinem Auftraggeber aufgefordert wurde. Dies ist für den Auftragnehmer nach dem MGET-Aufruf aus dem Transaktionsstatus ablesbar.

Beispiel 2: Nachricht und Prepare an Auftragnehmer und PEND KP

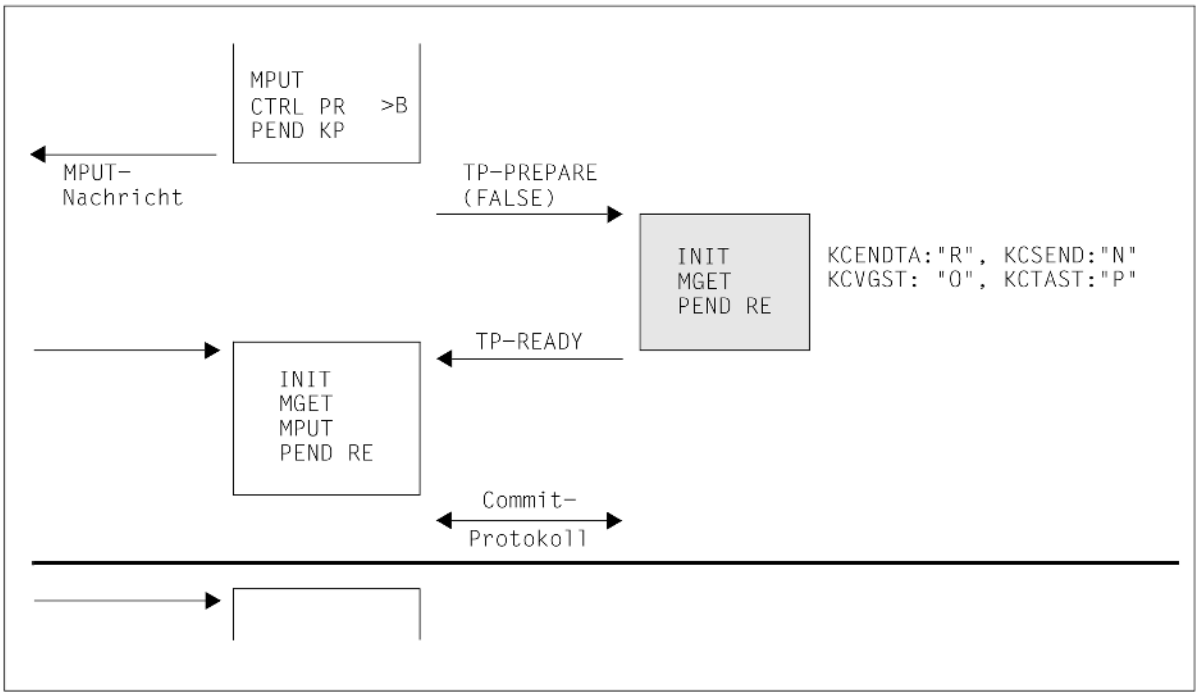


Dieser Fall ist bei der Kommunikation über das OSI TP-Protokoll als Normalfall anzusehen, da die Reihenfolge der KDCS-Aufrufe dem Ablauf des Protokollflusses am besten entspricht. Außerdem liegt hier nach Transaktionsende die Kontrolle im Auftraggeber-Vorgang, wodurch ein Vorgangs-Wiederanlauf erleichtert wird.

In dem zweiten Teilprogrammlauf des Auftraggebers kann in einem Dialog-Vorgang statt des PEND SP auch ein MPUT zum Client und ein PEND RE abgesetzt werden. Der Ablauf sieht dann wie folgt aus:



Beispiel 3: Keine Nachricht an den Auftragnehmer und CTRL PR



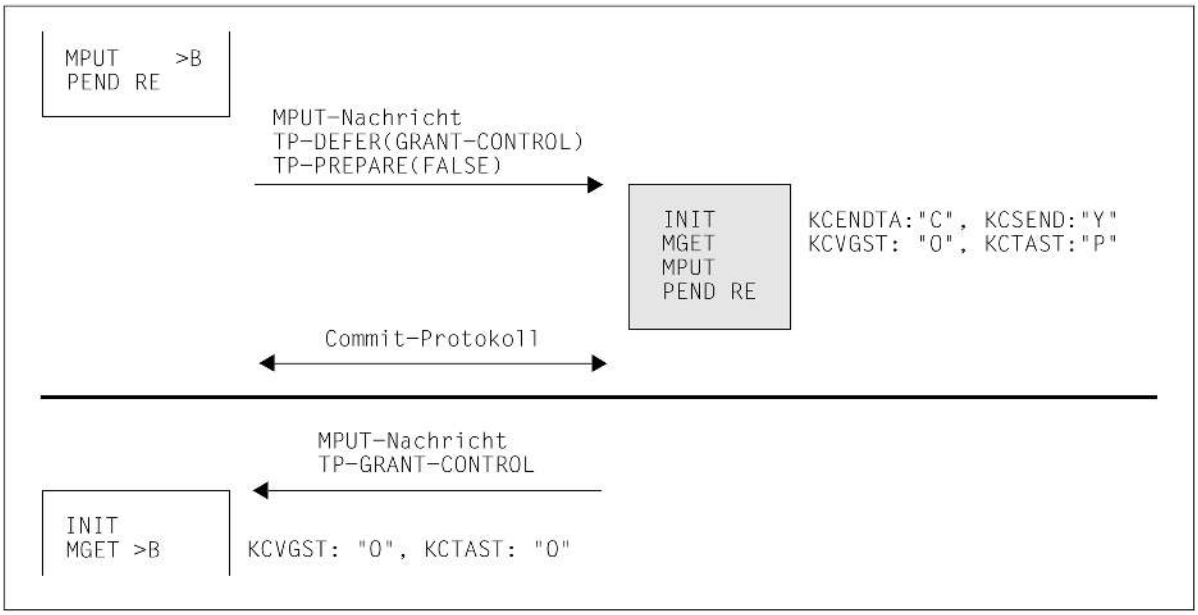
In diesem Fall erhält der Auftragnehmer nur eine Aufforderung zum Transaktionsende, aber keine Nachricht vom Auftraggeber. Das Senderecht wechselt deshalb nicht zum Auftragnehmer (DATA-PERMITTED=FALSE) und liegt zum Transaktionsende - wie auch in Beispiel 2 - beim Auftraggeber.

Statt des MPUT, PEND KP im ersten Teilprogrammlauf kann auch nur ein PEND PA/PR abgesetzt werden. Wenn der Auftraggeber-Vorgang ein Asynchron-Vorgang ist, ist nur diese zweite Variante möglich.

Beachten Sie, dass zum Start des Teilprogrammlaufs nach dem PEND KP nur auf die Nachricht vom Client gewartet wird, nicht aber auf den TP-READY des Auftragnehmers. D.h. nach PEND PA/PR wird, sofern nicht auf eine DGET-Nachricht gewartet wird, das Folge-Teilprogramm sofort gestartet.

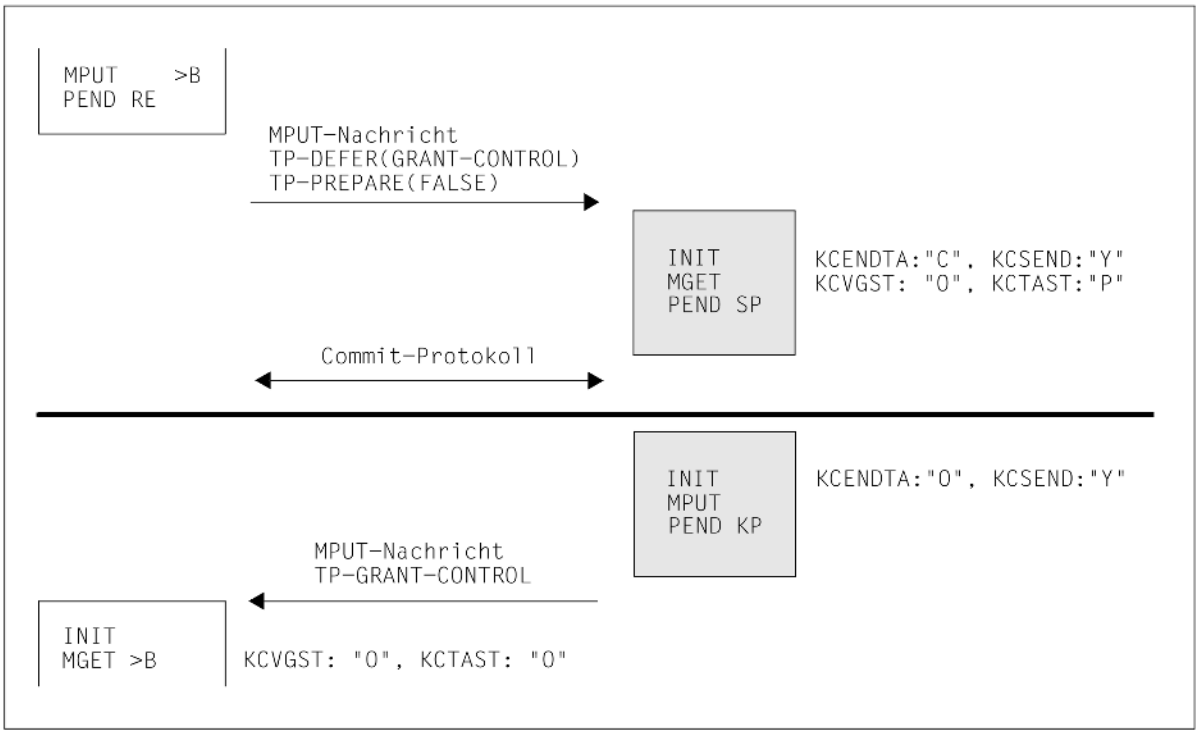
Statt des MPUT, PEND RE im zweiten Teilprogrammlauf kann auch nur ein PEND SP abgesetzt werden. Wenn der Auftraggeber-Vorgang ein Asynchron-Vorgang ist, ist nur diese zweite Variante möglich.

Beispiel 4: Nachricht an Auftragnehmer und PEND RE

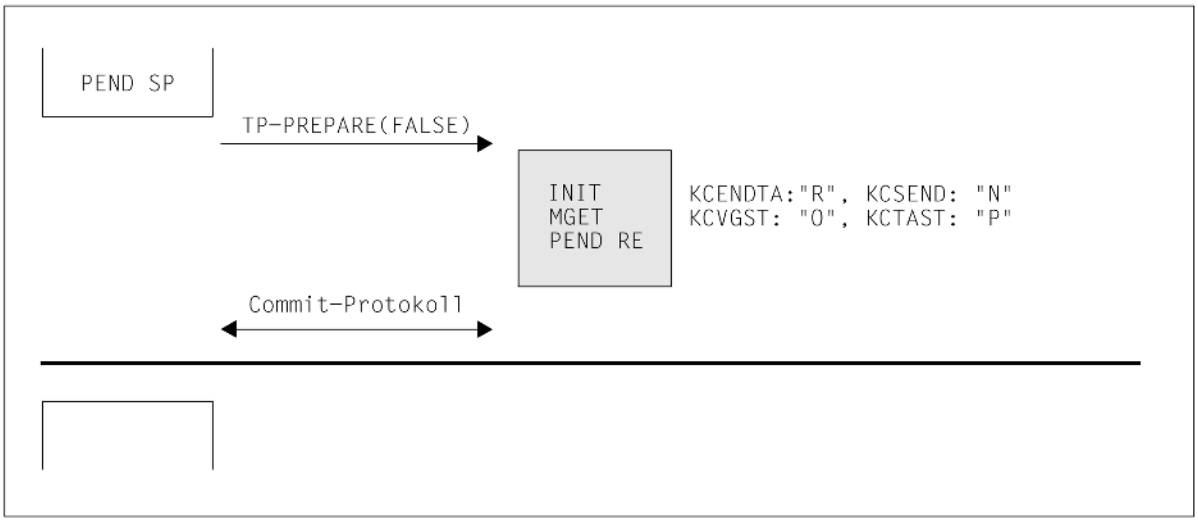


Dieser Fall kommt der Situation bei Verwendung des LU6.1 Protokolls am nächsten. Insbesondere sind hier für den Auftragnehmer die Vorgangs- und Transaktionsstati identisch zu denen bei der Kommunikation über das LU6.1 Protokoll, so dass sich diese Programme hier unverändert einsetzen ließen. Zu beachten ist allerdings, dass diese Programme die für die LU6.1-Kommunikation empfohlene Bottom-Up Strategie nicht einhalten.
In diesem Anwendungsfall liegt die Kontrolle bzw. das Senderecht zum Transaktionsende beim Auftragnehmer.

Der Auftragnehmer hat in diesem Fall auch die Möglichkeit, an Stelle des PEND RE einen PEND SP-Aufruf zu verwenden. Der Ablauf ändert sich dann wie folgt:

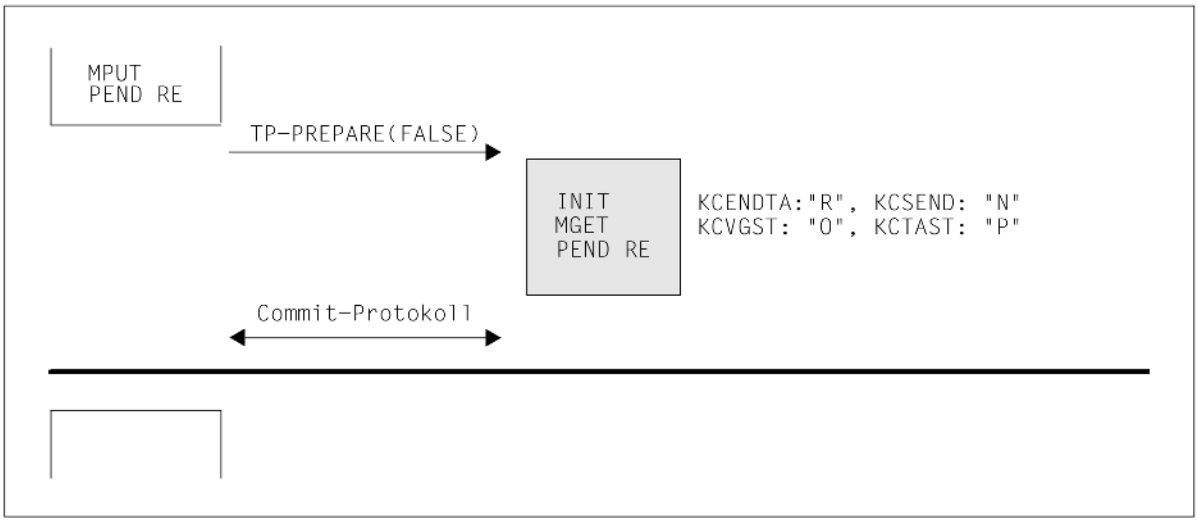


Beispiel 5: Keine Nachricht an Auftragnehmer und PEND SP/RE

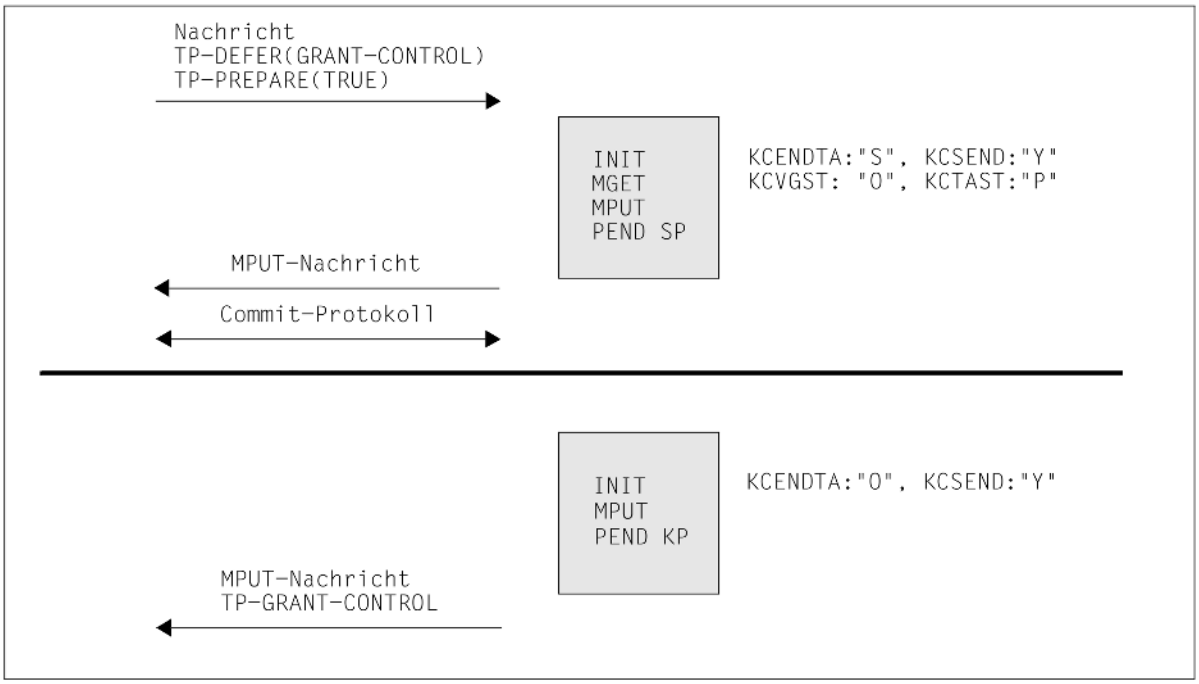


Dieses Beispiel zeigt, dass die Auftragnehmer-Vorgänge in alle Transaktionen miteingeschlossen sind, auch wenn innerhalb der letzten Transaktion keine Kommunikation zwischen dem Auftraggeber und dem Auftragnehmer stattgefunden hat. Bei Nutzung des OSI TP-Protokolls mit Chained Transactions gibt es keine so genannten lokalen Transaktionen. Dies müssen Sie beim Design von verteilten Anwendungen beachten.

Im Auftraggeber kann statt des PEND SP auch ein MPUT zum Client und ein PEND RE abgesetzt werden. Der Ablauf sieht dann wie folgt aus:



Beispiel 6: Defer-Grant-Control und Prepare(True)

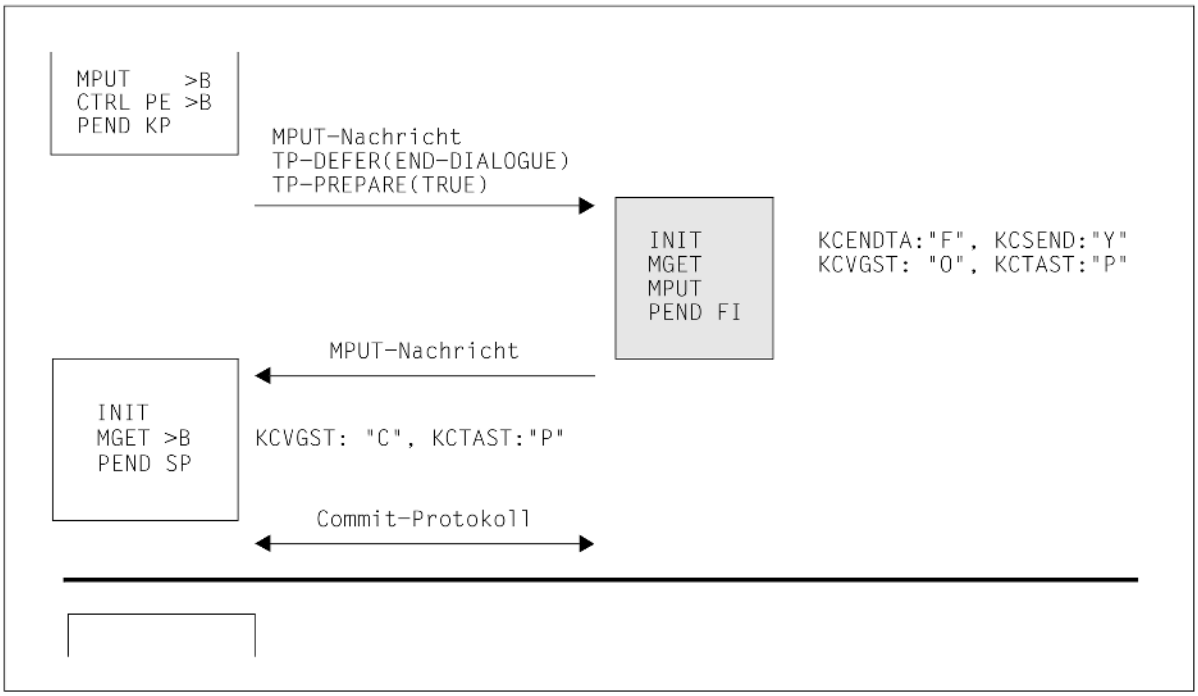


Dies ist ein exotischer Fall, der nur bei einer heterogenen Kopplung auftreten kann. Der Auftragnehmer ist hier in der Situation zwei Nachrichten nacheinander an den Auftraggeber senden zu müssen. Die erste Nachricht muss noch innerhalb der aktuellen Transaktion gesendet werden. Nach Transaktionsende bleibt das Senderecht beim Auftragnehmer und damit kommt die erste Nachricht der Folge-Transaktion ebenfalls vom Auftragnehmer.

Dialogende

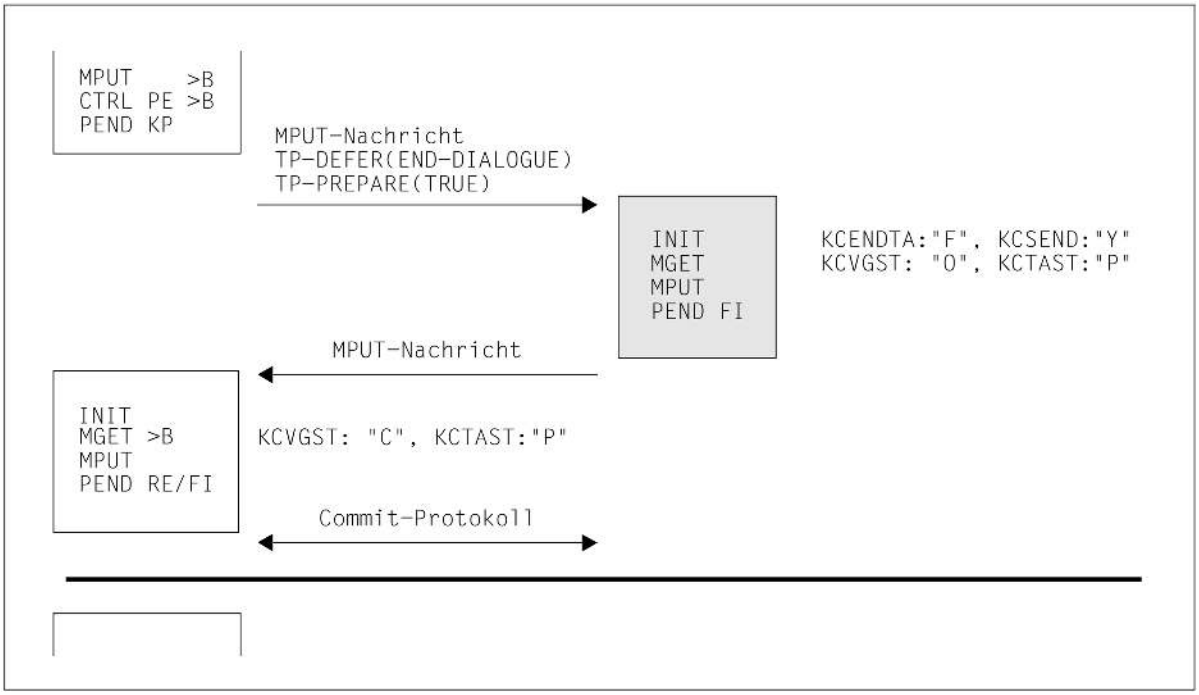
Bei Nutzung der Commit Funktionalität kann ein Auftragnehmer nur dann den Vorgang beenden, wenn ihn sein Auftraggeber dazu aufgefordert hat.
Im Normalfall werden die Auftragnehmer-Vorgänge zuerst beendet; anschließend kann sich der Auftraggeber-Vorgang beenden. Auch ein gleichzeitiges Beenden des Auftragnehmer- und Auftraggeber-Vorgangs ist möglich. Soll der Auftraggeber-Vorgang weitergeführt werden, dann muss der Auftragnehmer mit dem Aufruf CTRL PE aufgefordert werden, seinen Vorgang zu beenden.

Beispiel 7: Nachricht und End-Dialogue an Auftragnehmer und PEND KP

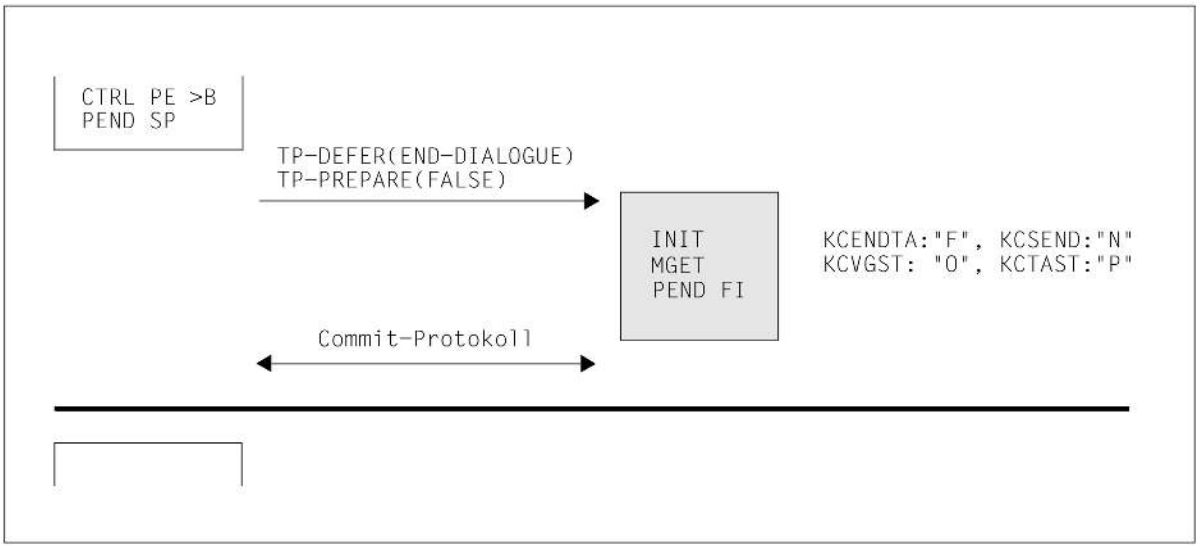


In diesem Beispiel kann der Auftragnehmer noch eine letzte Nachricht an den Auftraggeber übermitteln, bevor er seinen Vorgang beendet.

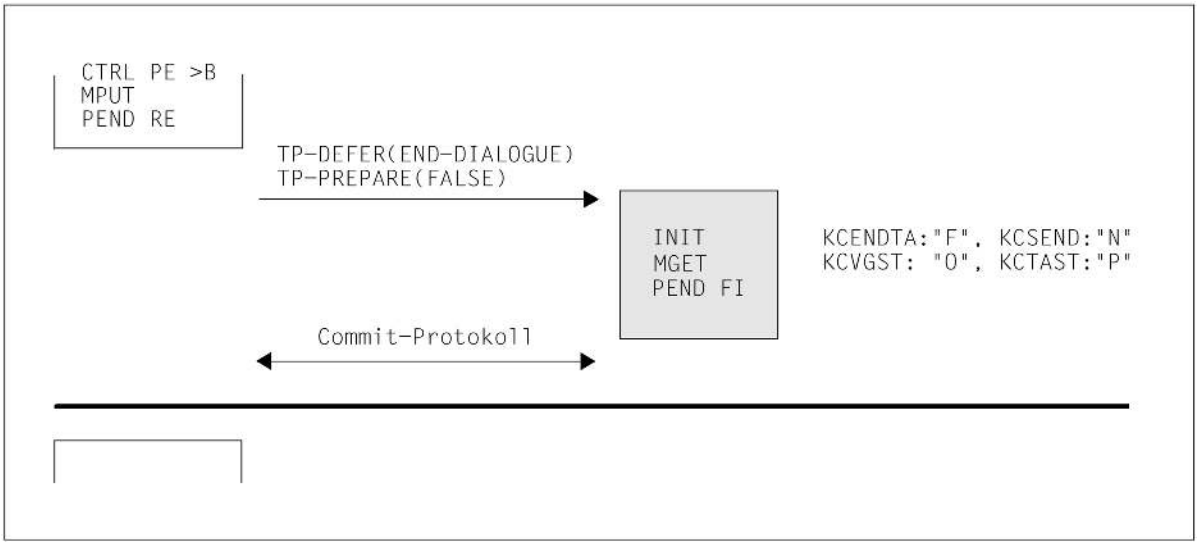
In dem zweiten Teilprogrammlauf des Auftraggebers kann statt des PEND SP auch ein MPUT zum Client und ein anderer PEND-Aufruf mit Transaktions- oder Vorgangsende-Anforderung gegeben werden. Der Ablauf sieht dann wie folgt aus:



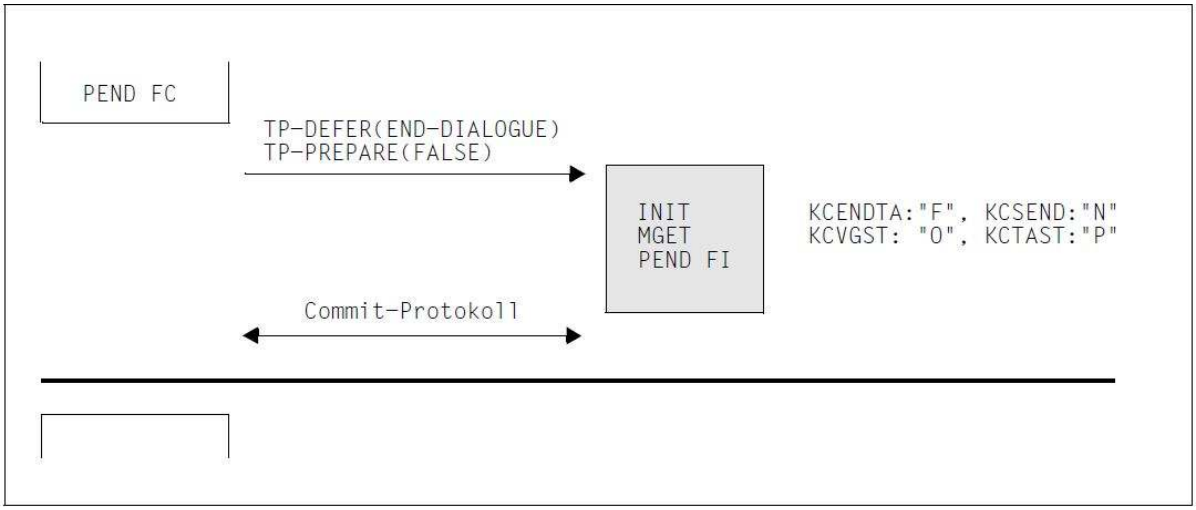
Beispiel 8: Keine Nachricht an den Auftragnehmer und PEND SP/RE



In dem ersten Teilprogrammlauf des Auftraggebers kann statt des PEND SP auch ein MPUT zum Client und ein PEND-Aufruf mit Transaktions- oder Vorgangsende-Anforderung gegeben werden. Es ergibt sich dann folgender Ablauf:

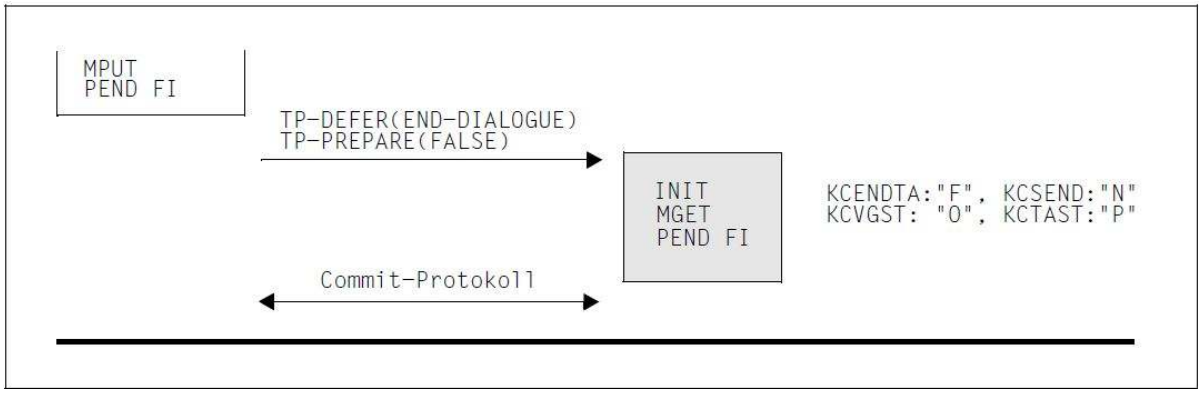


Beispiel 9: Keine Nachricht an den Auftragnehmer und PEND FC/FI

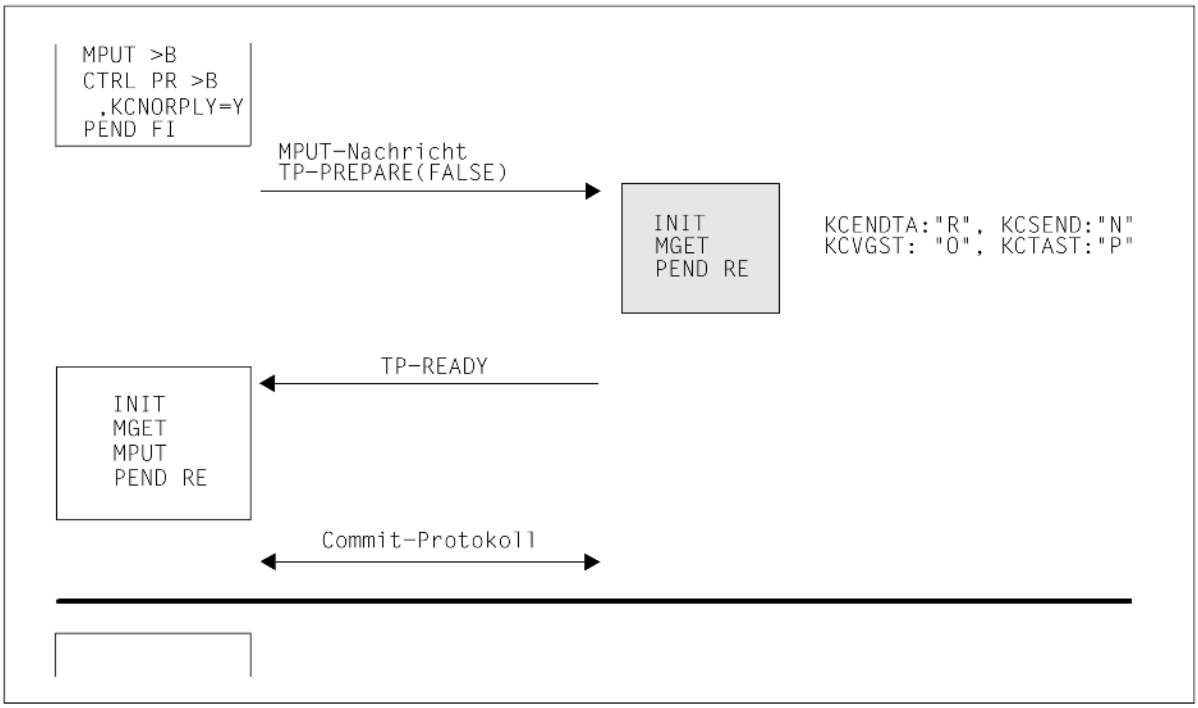


Im hier dargestellten Fall soll der Auftraggeber-Vorgang nicht wie in den Beispielen 6 und 7 fortgeführt werden, sondern beendet sich gleichzeitig mit dem Auftragnehmer. Bei PEND FC (Vorgangs-Kettung) wird der Dialog-Schritt in einem Folgevorgang fortgesetzt.

In dem ersten Teilprogrammlauf des Auftraggebers kann statt des PEND FC auch ein MPUT zum Terminal und ein PEND FI abgesetzt werden; in diesem Fall entfällt das Folge-Vorgang auf der Auftraggeberseite. Der Ablauf sieht dann folgendermaßen aus:



Beispiel 10: Nachricht an den Auftragnehmer und CTRL PR, KCNORPLY=Y



In diesem Fall erhält der Auftragnehmer eine Aufforderung zum Transaktionsende und eine Nachricht vom Auftraggeber, ohne dass der Auftraggeber das Senderecht an den Auftragnehmer abgibt (DATA-PERMITTED=FALSE). Der Auftragnehmer darf keine Nachricht mehr senden. Der Auftraggeber wartet mit dem Start des Folge-Teilprogramms nach dem PEND KP auf den Empfang des TP-READY.

5.4.9.2 Mehrere Auftragnehmer

Die Situation für einen Auftraggeber ist bei einer Kommunikation mit mehr als einem Auftragnehmer im Wesentlichen unabhängig von der konkreten Anzahl von Auftragnehmern. Deshalb genügt es bei Betrachtung dieser Konstellation von einem Auftraggeber (A) und zwei Auftragnehmern (B und C) auszugehen.

Aus Sicht der Auftragnehmer ist dieser Fall identisch zu den im vorigen Abschnitt geschilderten Situationen, da die Auftragnehmer als einzigen Kommunikationspartner den Auftraggeber kennen.

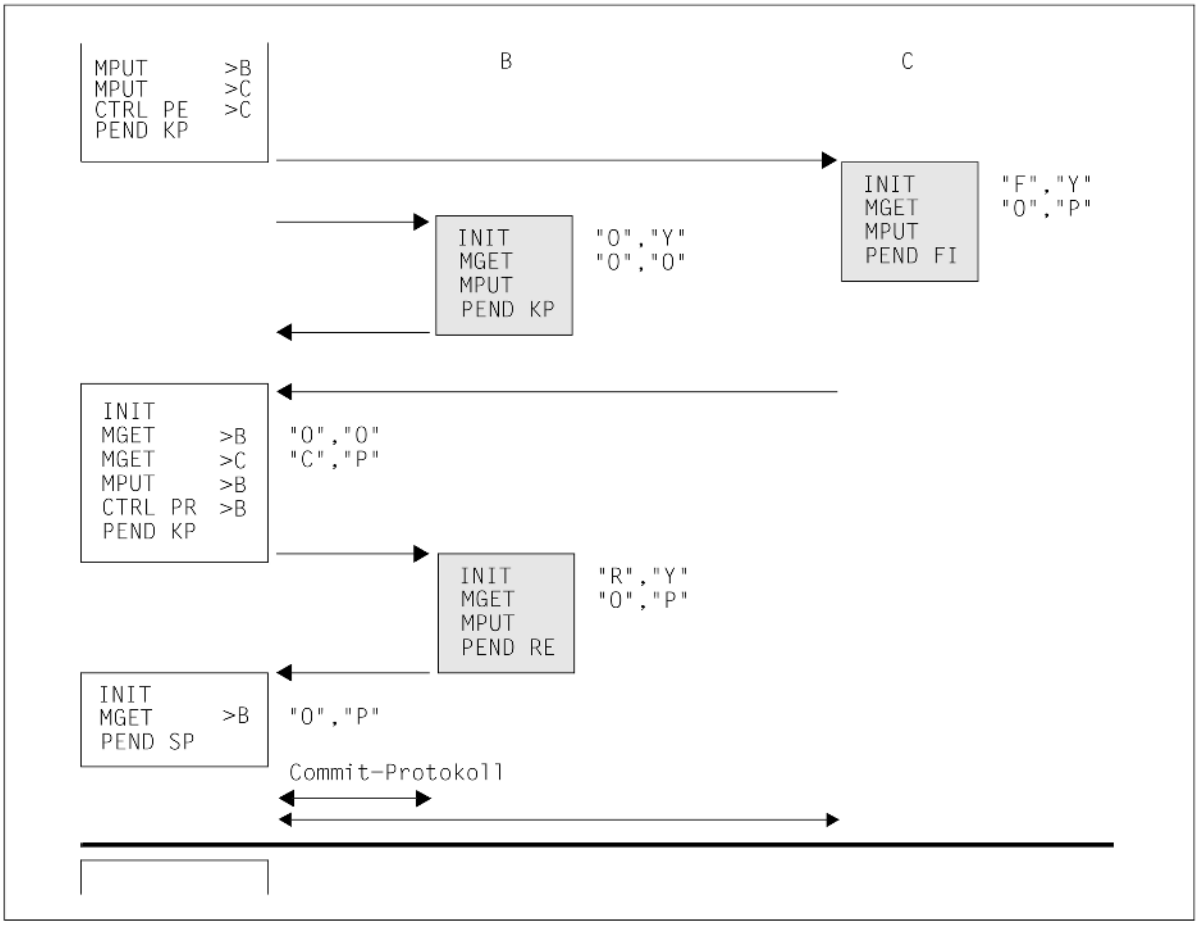
Aber auch für den Auftraggeber bringt dieses Szenario keine wesentlichen neuen Aspekte. Allerdings erhöht sich für ihn die Anzahl der möglichen Kombinationen.

Mit jedem einzelnen Auftragnehmer kann der Auftraggeber ebenso kommunizieren wie im vorigen Abschnitt dargestellt. Hinzu kommt, dass er in einem Verarbeitungsschritt entweder nur mit einem oder aber mit mehreren Auftragnehmern kommunizieren kann. Der Folge-Teilprogrammlauf im Auftraggeber-Vorgang wird immer erst dann gestartet, wenn von allen Auftragnehmern, an die im letzten Verarbeitungsschritt Nachrichten gesendet wurden, Antworten eingetroffen sind.

Bezüglich Transaktions- und Dialogende kann der Auftraggeber entweder mit Hilfe des Aufrufs CTRL gezielt einzelne Auftragnehmer dazu veranlassen, Transaktions- bzw. Dialogende anzufordern, oder aber er kann alle Auftragnehmer gleichzeitig von dieser Situation unterrichten, indem er einen entsprechenden PEND-Aufruf verwendet.

Da sich also die Situation nicht wesentlich geändert hat gegenüber der Kommunikation mit einem Auftragnehmer soll hier ein Beispiel genügen. Aus Platzgründen wird dabei auf die Darstellung des Protokollflusses verzichtet.

Beispiel 11: Mehrere Auftragnehmer



In dem Beispiel kommuniziert Auftraggeber A mit den Auftragnehmern B und C. Der Dialog mit C soll abgeschlossen werden, aber C soll noch eine letzte Nachricht an A senden. Dazu verwendet A einen MPUT sowie einen CTRL PE an Partner C. Mit Partner B soll der Dialog noch nicht beendet werden, daher wird ihm nur eine Nachricht übermittelt und der Programmlauf wird von A mit PEND KP abgeschlossen, um die Transaktion offen zu halten.

C erkennt an den Anzeigen "F,Y", dass er an A noch eine Nachricht senden muss und dass die Transaktion mit PEND FI zu beenden ist. Für B bleiben Transaktion und Dialog noch offen, angezeigt durch "O,Y".

Im zweiten Teilprogrammlauf fordert A mit einem CTRL PR nun auch B auf, die Transaktion zu beenden. A möchte jedoch noch in der aktuellen Transaktion die Antwort von B erhalten und beendet den Programmlauf deswegen mit PEND KP.
B erhält die Transaktionsendeanforderung angezeigt durch die Stati "R,Y". B sendet eine Antwort an A und beendet für sich die Transaktion mit PEND RE.

Nachdem nun sowohl C als auch B Transaktionsende angefordert haben, kann A die verteilte Transaktion endgültig beenden. Mit dem Transaktionsende wird gleichzeitig der Dialog mit C beendet, der Dialog mit B hingegen bleibt bestehen.

5.4.9.3 Komplexere Dialogbäume

Abschließend sollen Fälle untersucht werden, in denen ein Vorgang (B) sowohl mit einem Auftraggeber (A) als auch mit einem Auftragnehmer (C) über das OSI TP-Protokoll mit Chained Transactions kommuniziert. Gegenüber den zuvor betrachteten Fällen ergibt sich hier nur für den Zwischenknoten B eine neue Situation. Er besitzt gleichzeitig einen Auftraggeber A und einen Auftragnehmer C.

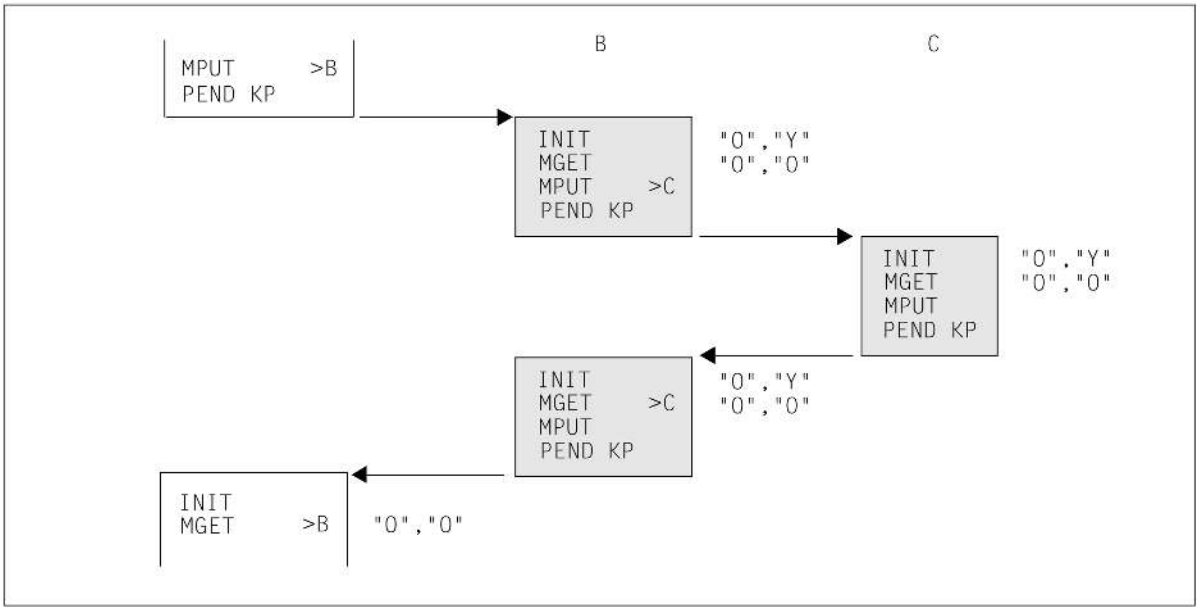
Bei Nutzung des OSI TP-Protokolls kann ein Zwischenknoten nicht frei darüber entscheiden, wann ein Transaktions- oder Dialogende erfolgen soll, auch nicht auf dem Dialog mit seinem Auftragnehmer. Der Zwischenknoten B kann seinen Auftragnehmer C erst auffordern, die Transaktion oder den Vorgang zu beenden, wenn ihn sein Auftraggeber zur Transaktionsbeendigung aufgefordert hat.

Im Folgenden werden Beispiele für einzelne charakteristische Situationen gegeben. Es bestehen zahlreiche weitere Möglichkeiten, die sich aus den in den vorigen Abschnitten angegebenen Fällen zusammensetzen lassen.

Datentransferphase

Die Datentransferphase ist dadurch gekennzeichnet, dass kein Partner CTRL-Aufrufe verwendet und dass die Programmläufe ausschließlich mit PEND KP beendet werden.

Beispiel 12: Datentransferphase bei mehrstufigen Transaktionsbäumen



B muss nicht immer abwechselnd mit A und mit C kommunizieren, so wie dies in diesem Beispiel dargestellt ist. Es ist auch möglich, dass B mehrere Dialog-Schritte hintereinander mit C abwickelt, oder aber für einige Zeit nur mit A kommuniziert und C erst später wieder an der Kommunikation beteiligt.

Wichtig ist jedoch, dass B nicht gleichzeitig Nachrichten an A und an C schicken darf. Ein Zwischenknoten darf das Senderecht entweder an seinen Auftraggeber oder aber an einen oder mehrere Auftragnehmer abgeben, aber nicht gleichzeitig an Auftraggeber und Auftragnehmer.

Ein Vorgang darf das Senderecht zum Transaktionsende auf höchstens einem Dialog abgeben.

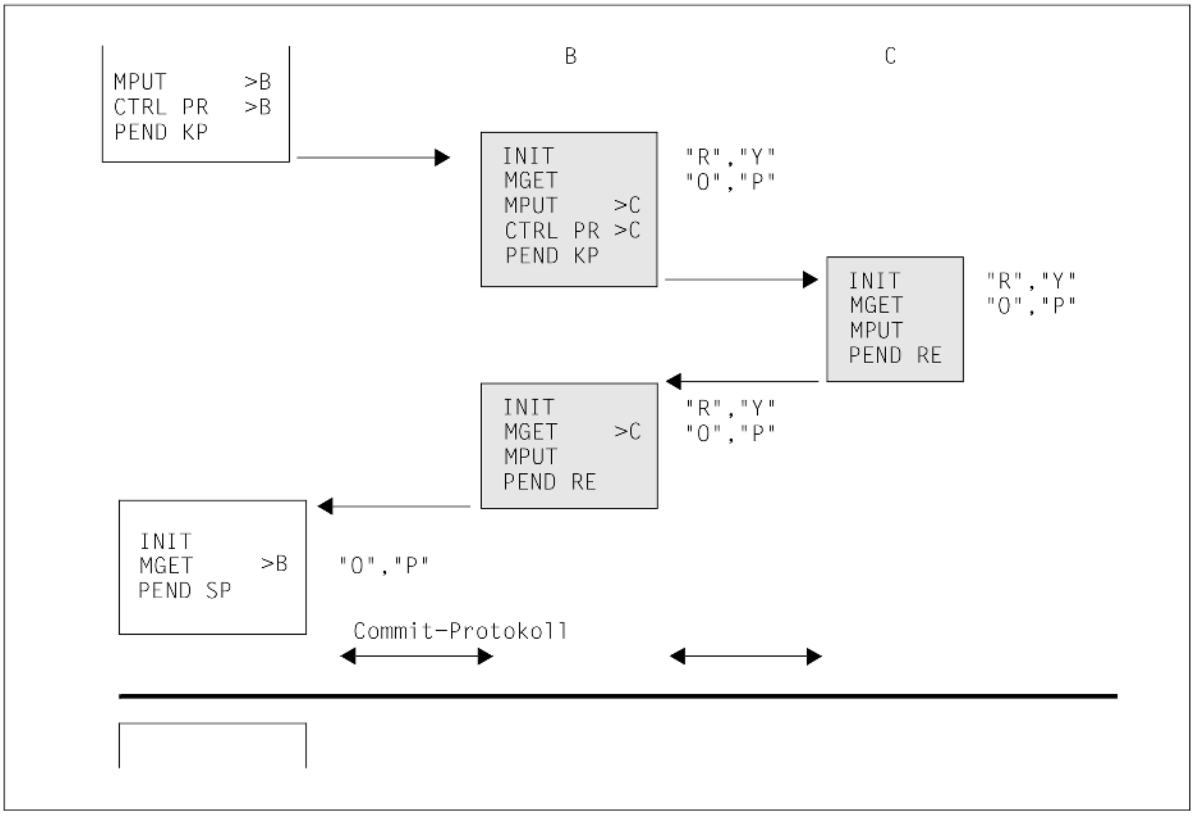
Transaktionsende

Nachdem B von A die Aufforderung zum Transaktionsende erhalten hat, bleiben B drei Möglichkeiten:

- B kann C eine Nachricht senden und C gleichzeitig auffordern, die Transaktion zu beenden (vgl. Beispiele 12,14, 15).
- B kann zunächst die Datentransferphase gegenüber C fortsetzen und C erst in einem späteren Verarbeitungsschritt auffordern, die Transaktion zu beenden (vgl. Beispiel 13).
- B kann darauf verzichten in der laufenden Transaktion noch weiter mit C zu kommunizieren und sofort selbst Transaktionsende anfordern (vgl. Beispiel 16).

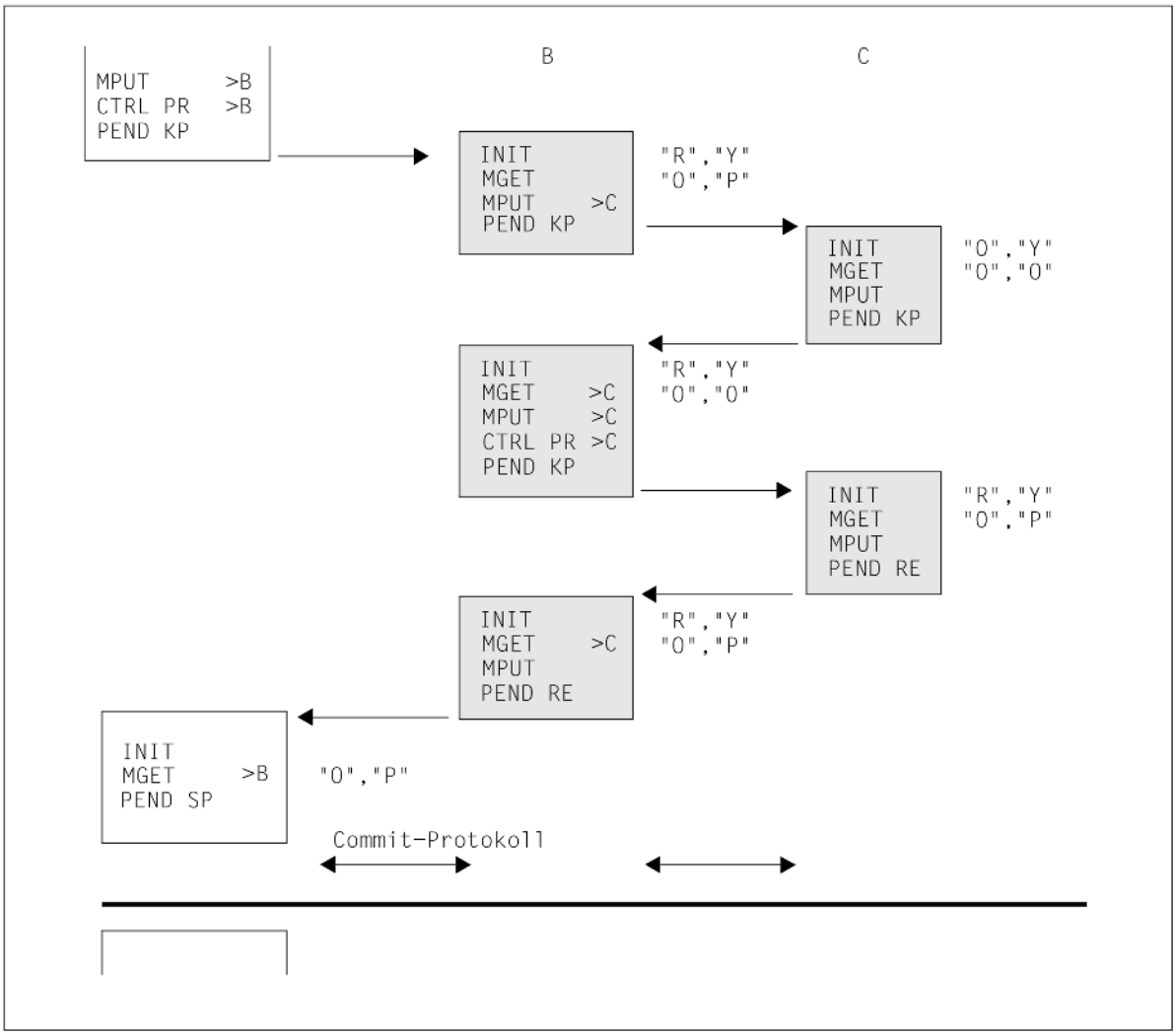
Das Senderecht zum Transaktionsende kann bei A, B oder C liegen.

Beispiel 13: Senderecht zum Transaktionsende liegt bei A



B kann in obigem Beispiel das Senderecht zum Transaktionsende nicht an C abgeben, da A das Senderecht zum Transaktionsende nicht an B abgegeben hat.

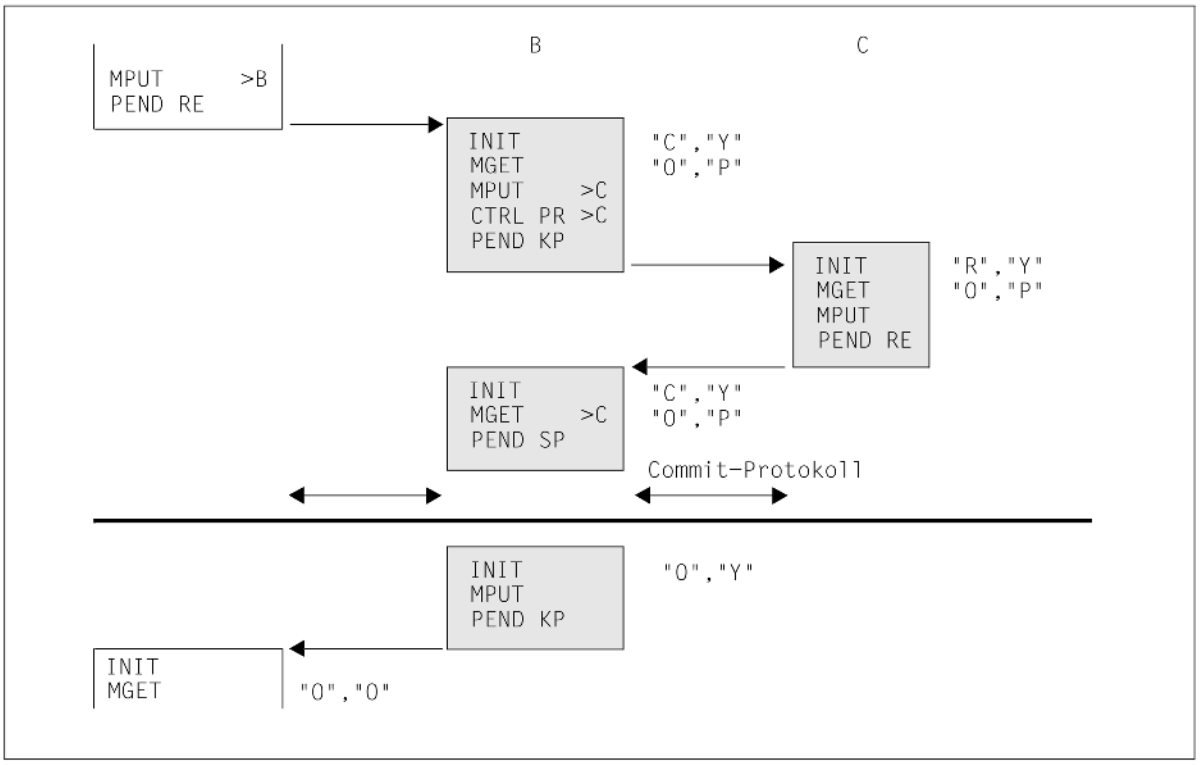
Beispiel 14: B setzt Dialog mit C zunächst fort - Senderecht zum Transaktionsende liegt bei A



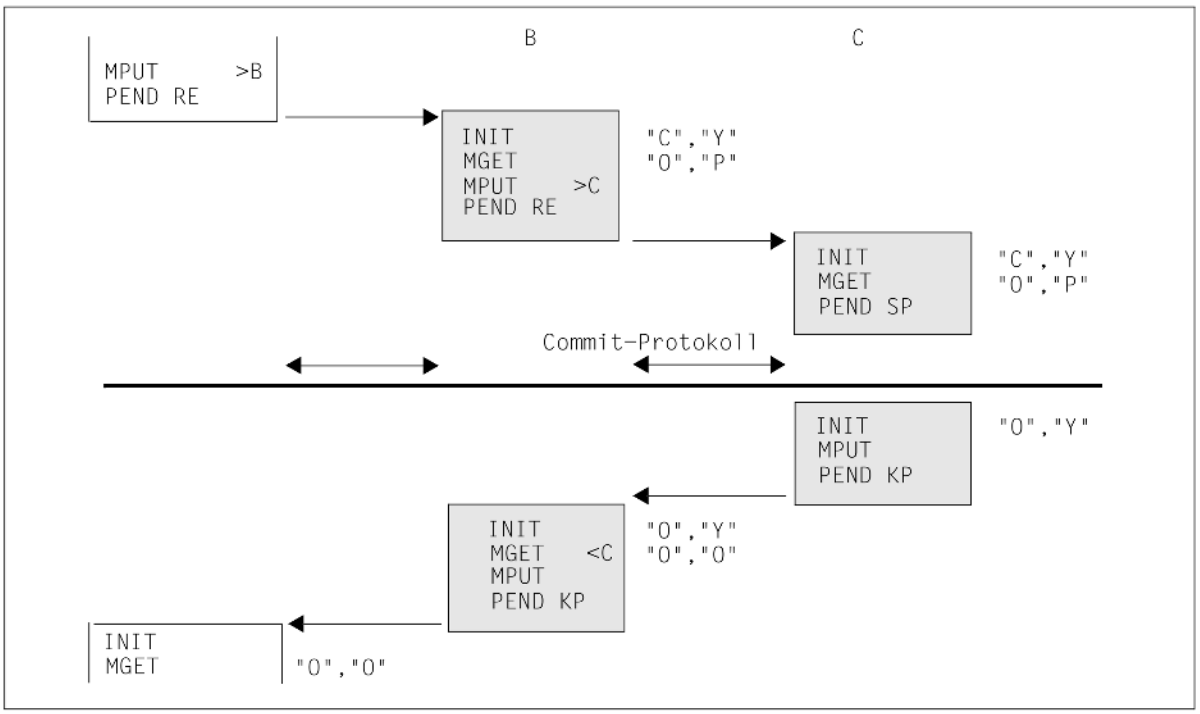
In diesem Beispiel verzichtet B zunächst darauf, C zum Transaktionsende aufzufordern, und setzt stattdessen die Datentransferphase gegenüber C fort. Im Beispiel wird nur noch eine Dialog-Nachricht ausgetauscht, die Datentransferphase könnte jedoch noch weiter fortgesetzt werden. Dabei dürfen B und C jedoch die Programmläufe nur mit PEND KP abschließen. Irgendwann muss B dann C auffordern, die Transaktion zu beenden.

Auch in diesem Beispiel kann B das Senderecht zum Transaktionsende nicht an C abgeben, da A das Senderecht zum Transaktionsende nicht an B abgegeben hat.

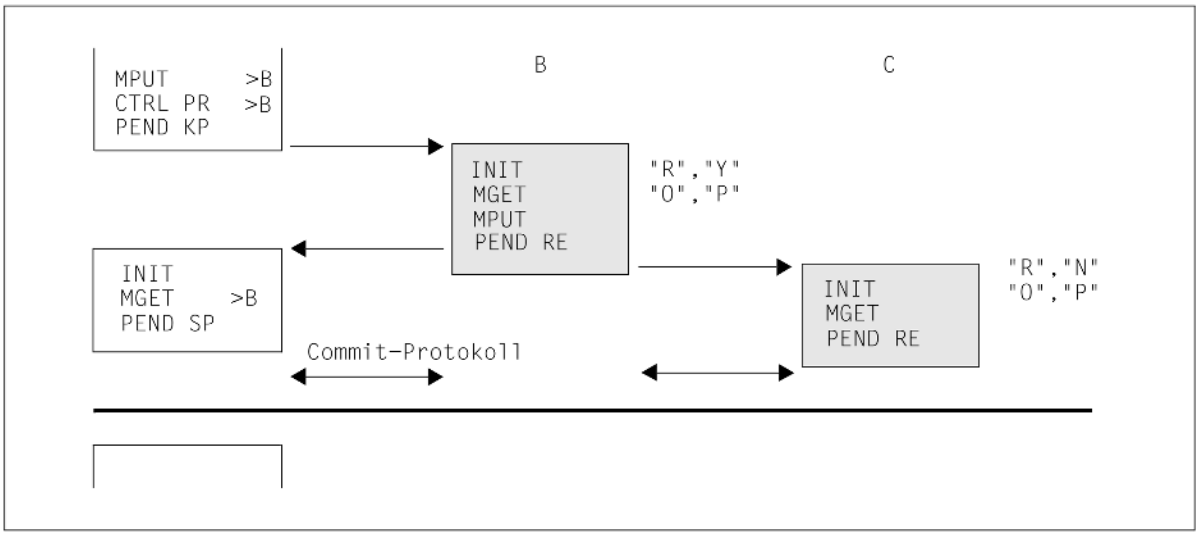
Beispiel 15: Das Senderecht zum Transaktionsende liegt bei B



Beispiel 16: Das Senderecht zum Transaktionsende liegt bei C



Beispiel 17: Vor Transaktionsende keine Nachricht an Auftragnehmer



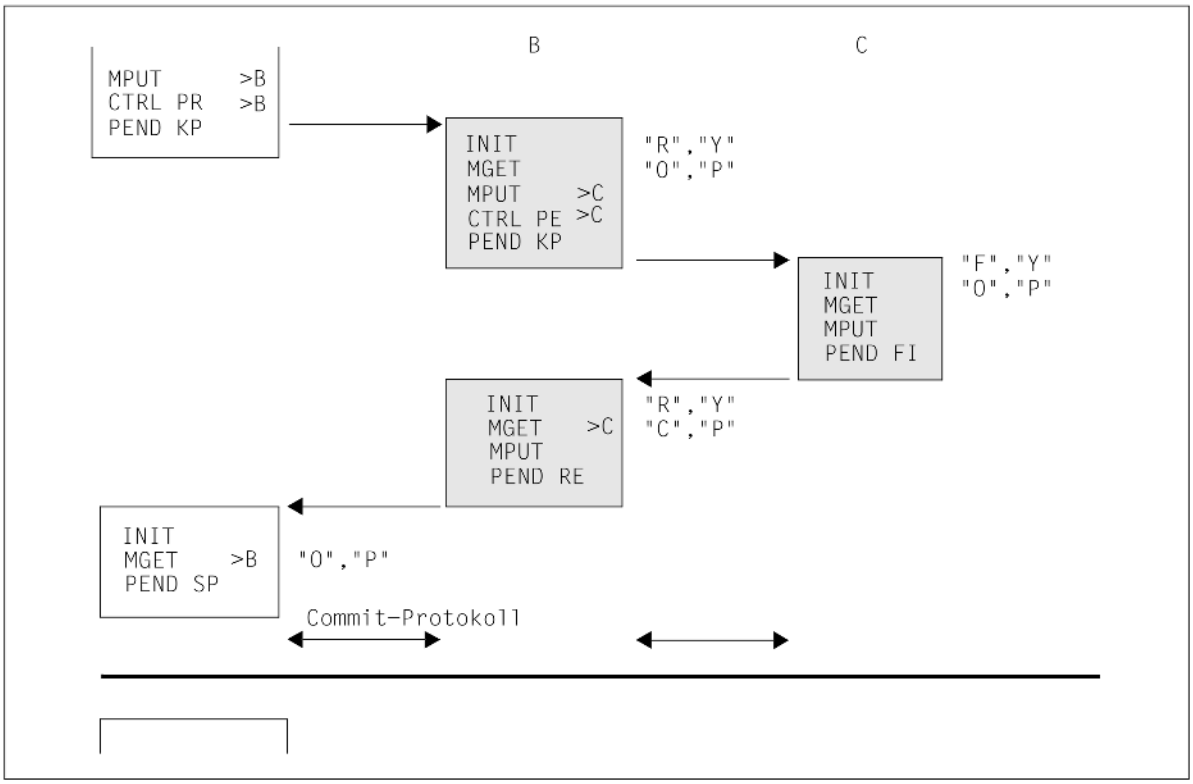
In diesem Beispiel verzichtet B darauf in der laufenden Transaktion noch eine Nachricht an C zu senden und fordert unmittelbar das Transaktionsende an.

Nach dem PEND RE von Knoten B wird A die MPUT-Nachricht zugestellt und an C wird ein PREPARE Protokollelement gesendet, mit dem C aufgefordert wird, das Transaktionsende einzuleiten. C erhält dabei keine Benutzernachricht mehr. Der MGET-Aufruf bei C dient nur dazu den Status des Dialogs mit B zu lesen; dieser Aufruf kann auch entfallen.

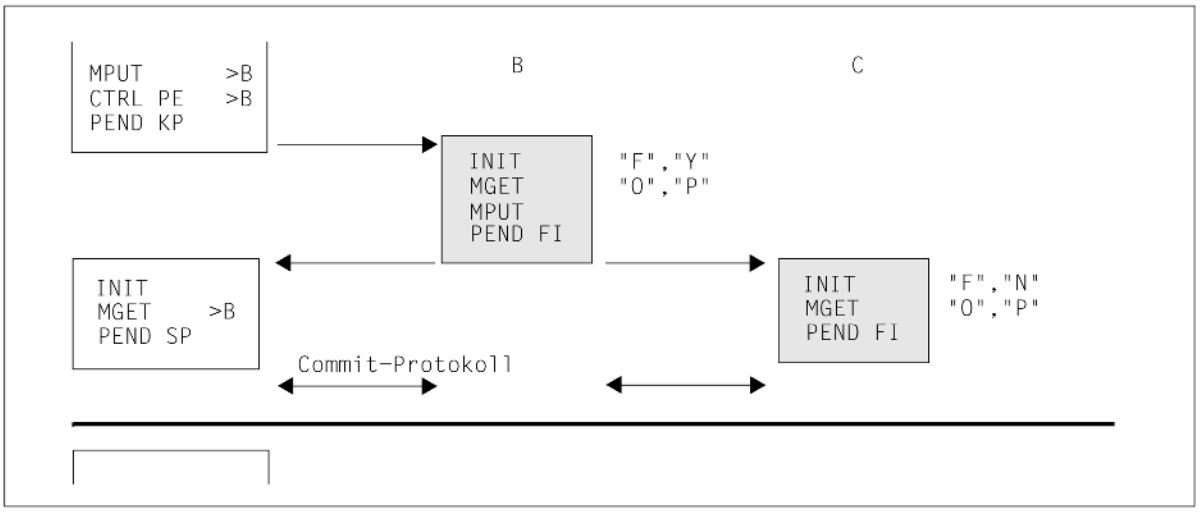
Dialogende

Zur Dialog-Beendigung durch den Auftraggeber bieten sich für einen Zwischenknoten dieselben Möglichkeiten wie zur Beendigung einer Transaktion. Diese sind in den beiden nachstehenden Beispielen erläutert.

Beispiel 18: B beendet zuerst den Dialog mit dem Auftragnehmer



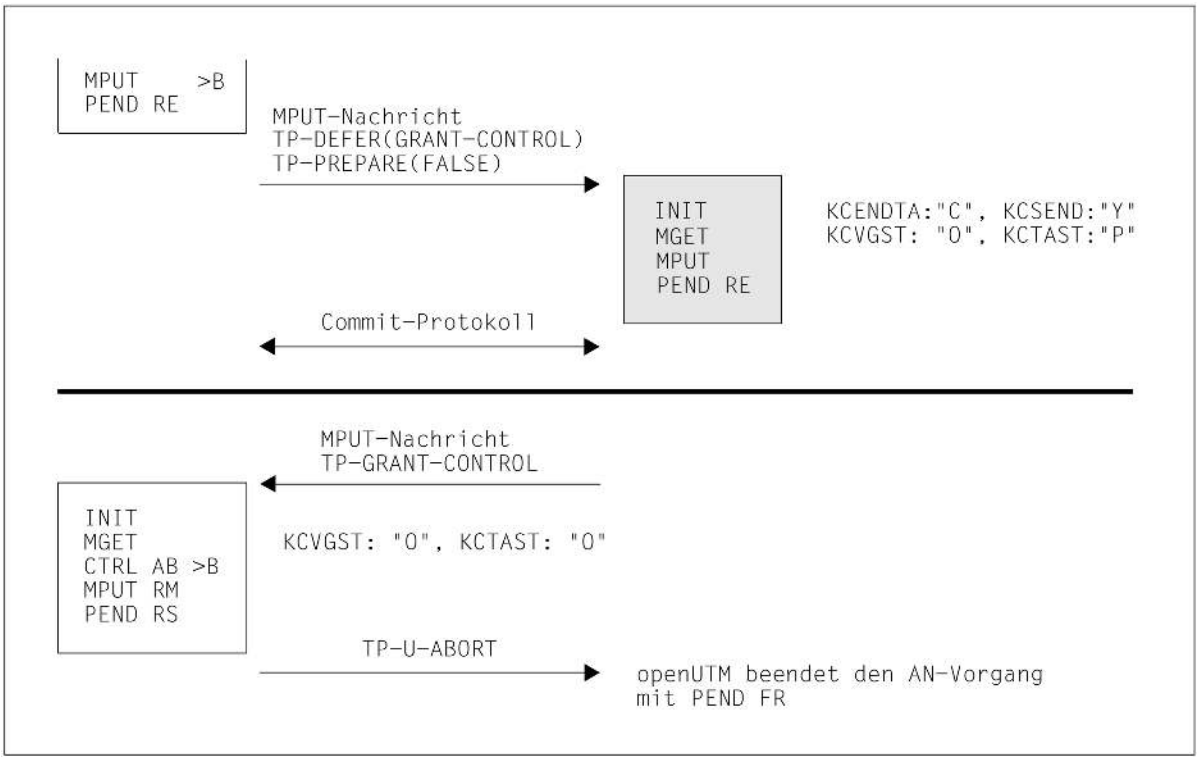
Beispiel 19: Gleichzeitiges Dialogende mit Auftragnehmer und Auftraggeber



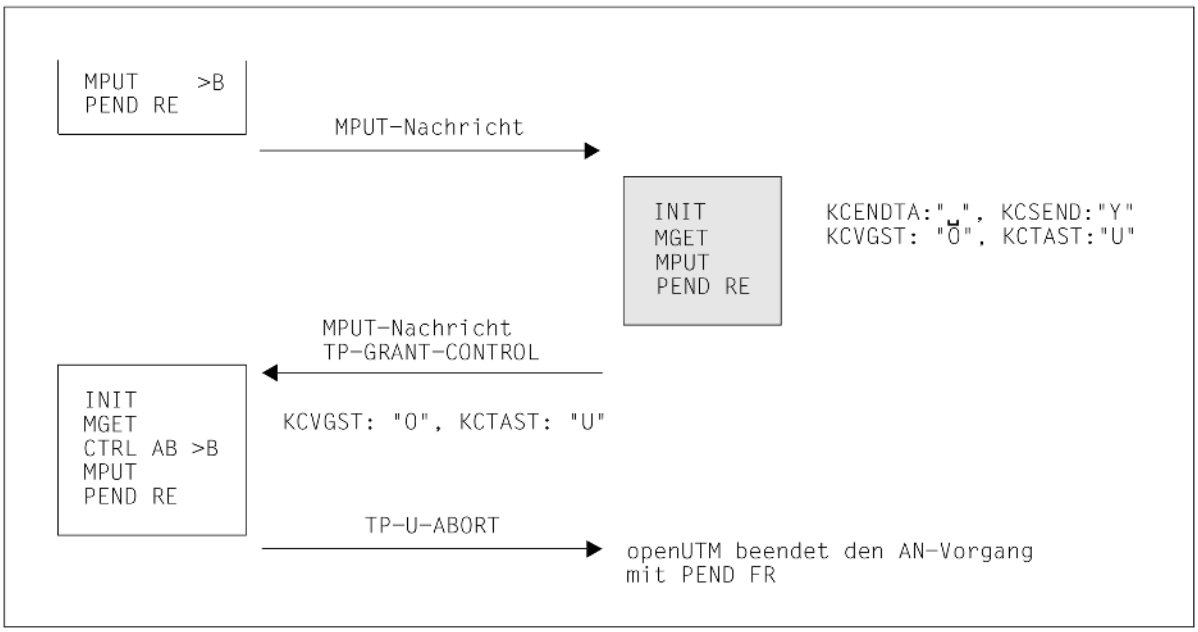
5.4.9.4 Beendigung eines Auftragnehmers mit CTRL AB

Mit dem CTRL AB wird ein Auftragnehmer-Vorgang abnormal beendet. Nach einem CTRL AB für einen Auftragnehmer, für den die Commit Funktionseinheit ausgewählt ist, muss der Auftraggeber die verteilte Transaktion zurücksetzen, bei einem Auftragnehmer-Dialog ohne Commit nicht.

Beispiel 20: Beendigung eines Dialogs, für den die Commit FU ausgewählt ist



*Beispiel 21: Beendigung eines Dialogs, für den die Commit FU **nicht** ausgewählt ist*



5.5 UTM-gesteuerte Queues bei verteilter Verarbeitung

Ein Auftraggeber-Vorgang kann per FPUT- oder DPUT-Aufruf einen Asynchron-Auftrag an einen fernen Asynchron-Vorgang richten (Remote Queuing). Der Auftraggeber-Vorgang kann dabei Dialog- oder Asynchron-Vorgang sein.

openUTM arbeitet bei Asynchron-Aufträgen an ferne Anwendungen mit zwei jeweils lokalen Queues: eine Queue liegt dabei in der Sender-Anwendung, die andere Queue beim Empfänger. Durch dieses Deferred Delivery-Prinzip ist das rechnerübergreifende Message Queuing mit openUTM vollständig unabhängig davon, ob gerade eine Verbindung besteht oder nicht: Falls keine Verbindung aufgebaut werden kann, bleibt der Auftrag solange in der lokalen Sender-Queue, bis eine Verbindung hergestellt ist. Nach Verbindungsaufbau gilt:

- Bei LU6.1 werden die Aufträge umgehend an den Partner übertragen.
- Bei OSI TP kann es noch eine bestimmte Zeit dauern, bis die Aufträge übertragen werden. Diese Zeit ist durch den in MAX CONRTIME generierten Wert beschränkt. Beachten Sie, dass bei CONRTIME=0 die Zeit auf 10 Minuten gesetzt wird.

Kommt es beim Senden eines Auftrags – also bei bestehender Verbindung - zu einem dauerhaften Fehler, so wird der Auftrag aus der lokalen Sender-Queue gelöscht, aber nicht in die entsprechende Message Queue der Partner-Anwendung eingetragen. In diesem Fall wird in der lokalen Anwendung eine Meldung K239 ausgegeben.

Zu einem dauerhaften Fehler, der zum Verlust des Auftrags führt, kommt es unter anderem, wenn der Auftrag an einen TAC gerichtet ist, der in der Partner-Anwendung gesperrt ist. Die genaue Ursache finden Sie in der Meldung K086 (LU6.1) oder K119 (OSI TP), die in der Partner-Anwendung ausgegeben wird.

Der Verlust des Auftrags kann verhindert werden, in dem für den LPAP- bzw. OSI-LPAP-Partner der lokalen Anwendung die Sicherung von Nachrichten in der lokalen Dead Letter Queue aktiviert wird, siehe Abschnitt „Dead Letter Queue“.

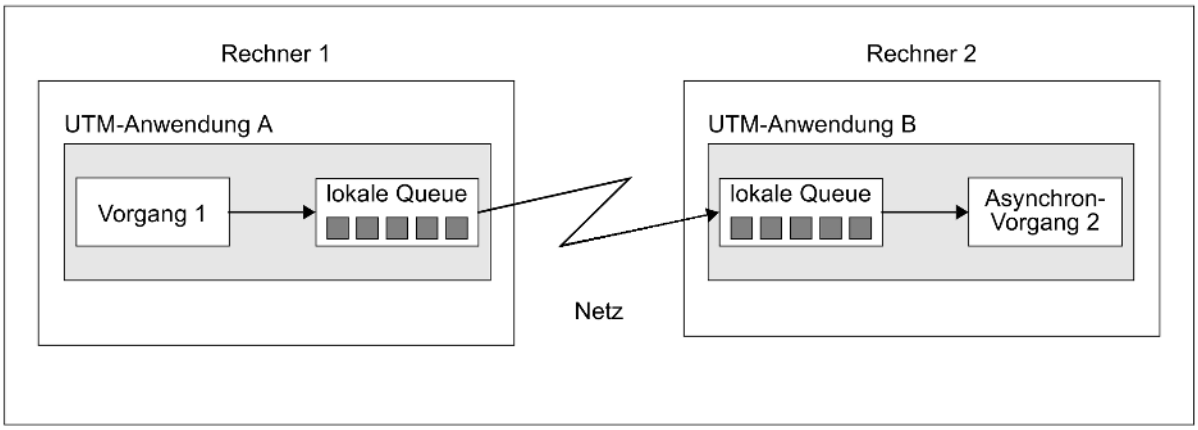


Bild: Remote Queuing mit openUTM

5.5.1 Auftraggeber-Seite

Der Auftragnehmer-Vorgang wird mit einem Aufruf **APRO AM** adressiert, dabei vergibt man im Feld KCPI die Vorgangs-Identifikation.

Bei verteilter Verarbeitung über OSI TP kann beim APRO-Aufruf gewählt werden, ob ein Asynchron-Auftrag mit globaler Transaktionssicherung übertragen werden soll oder nicht. Bei globaler Transaktionssicherung garantiert openUTM, dass der Auftrag genau einmal übertragen wird, sofern er nicht durch einen dauerhaften Fehler beim Senden (siehe "UTM-gesteuerte Queues bei verteilter Verarbeitung") verloren geht. Asynchron-Aufträge ohne globale Transaktionssicherung werden u.U. bei einem Verbindungsverlust mehrfach übertragen.

Nach dem APRO AM-Aufruf kann der Auftraggeber-Vorgang:

- mit FPUT einen Asynchron-Auftrag oder mit DPUT einen zeitgesteuerten Asynchron-Auftrag an den fernen Vorgang senden, wobei als Ziel in KCRN die Vorgangs-Identifikation anzugeben ist oder
- mit MCOM BC den Beginn eines Auftrags-Komplexes definieren und innerhalb des Komplexes mit DPUT einen Asynchron-Auftrag (Basisauftrag) an die Partner-Anwendung senden sowie zugehörige positive oder negative Quittungsaufträge erzeugen. Die Quittungsaufträge werden von der lokalen Anwendung bearbeitet.

  Bei MCOM BC definieren Sie die Komplex-Identifikation und tragen dabei in KCRN die Vorgangs-Identifikation ein. Bei DPUT muss dann in KCRN die Komplex-Identifikation angegeben werden.

Sie **müssen** innerhalb des Teilprogramms, das den fernen Vorgang mit APRO AM adressiert, einen FPUT- oder DPUT-Aufruf mit dieser Vorgangs-Identifikation absetzen, sonst bricht openUTM beim PEND den Vorgang mit KCRCCC=86Z ab und gibt die Vorgangs-Identifikation frei.

Die Vorgangs-Identifikation wird im Auftraggeber-Vorgang in folgenden Fällen freigegeben:

- nach einem erfolgreichem Aufruf FPUT NE bzw. DPUT NE
- beim folgenden PEND-Aufruf (auch bei PEND KP und PEND PA/PR)
- nach einem RSET-Aufruf
- nach dem Returncode 40Z nach einem FPUT- oder DPUT-Aufruf
- bei Auftrags-Komplexen mit dieser Vorgangs-Identifikation: Beim MCOM EC-Aufruf oder nach einem Returncode 40Z nach MCOM BC- oder nach einem DPUT-Aufruf.

Nach der Freigabe kann diese Vorgangs-Identifikation im Auftraggeber-Vorgang für eine neue Auftraggeber-/Auftragnehmer-Beziehung verwendet werden.

Der Auftrag auf der Auftraggeber-Seite wird aus der Message Queue gelöscht, wenn er erfolgreich übertragen und in die entsprechende Message Queue der Partner-Anwendung eingetragen wurde. Bei einem Nachrichten-Komplex wird - je nach Übertragungsergebnis - der positive oder negative Quittungsauftrag gestartet.

5.5.2 Auftragnehmer-Seite

Ein Asynchron-Auftrag an eine Partner-Anwendung wird dort so behandelt, als sei er in einem Vorgang in der eigenen Anwendung erzeugt worden: Asynchron-Aufträge aus Vorgängen der lokalen Anwendung und Asynchron-Aufträge, die aus fernen Vorgängen abgesetzt wurden, stehen in einer gemeinsamen Message Queue, die dem Asynchron-TAC zugeordnet ist. Für jeden Auftrag wird, sobald er an der Reihe ist und die Betriebsmittel zur Verfügung stehen, ein entsprechender Asynchron-Vorgang gestartet. Am Eintrag im Feld KCTERMN im KB-Kopf kann der Asynchron-Vorgang erkennen, ob der Auftraggeber ein ferner Vorgang war.

Asynchron-Vorgänge für das Remote Queuing werden genauso aufgebaut wie beim Locale Queuing (siehe "Aufbau eines Asynchron-Vorgangs"). Bei verteilter Verarbeitung über OSI TP gibt es jedoch eine weitere Möglichkeit: Asynchron-Aufträge an Dialog-Vorgänge.

Asynchron-Aufträge an ferne Dialog-Vorgänge (nur über OSI TP)

Bei einem Asynchron-Auftrag, für den das OSI TP-Protokoll verwendet wird und für den beim APRO globale Transaktionssicherung vereinbart wurde, kann der Empfänger des Auftrags auch ein **Dialog**-Vorgang sein.

Nach Empfang eines Asynchron-Auftrags für einen Dialog-Vorgang wird der Vorgang in der Partner-Anwendung sofort gestartet und nicht wie ein Auftrag an einen Asynchron-Vorgang zunächst in eine Message Queue eingetragen. In der Anwendung des Auftraggebers wird der Auftrag erst dann aus der Message Queue gelöscht, wenn der Dialog-Vorgang beendet ist. Bei einem Nachrichten-Komplex wird dann - je nach Verarbeitungsergebnis der positive oder negative Quittungsauftrag gestartet.

Ein Dialog-Vorgang, der durch einen Asynchron-Auftrag gestartet wird, muss die Transaktion mit PEND FI beenden und darf keinen MPUT an den Auftraggeber enthalten (KCENDTA=F und KCSEND=N).

5.6 Service-gesteuerte Queues bei verteilter Verarbeitung

Über LU6.1 und OSI TP können auch Nachrichten an TAC-Queues ferner Anwendungen ausfallsicher übertragen werden. Dabei geht man beim Generieren und Programmieren analog vor wie beim Senden von Aufträgen an Dialog- oder Asynchron-TACs in fernen Anwendungen (siehe "Adressierung ferner Vorgänge").

Generierung

In einer LTAC-Anweisung wird einem lokalen LTAC-Namen als RTAC-Name der Name einer TAC-Queue in der fernen Anwendung zugeordnet.

`LTAC REMOTEQ, RTAC=`*Name-der-TAC-Queue-in-ferner-Anwendung*

Programmierung

Die Nachricht wird mit einem APRO AM Aufruf adressiert und mit einem nachfolgendem FPUT-Aufruf gesendet.

```
INIT
...
APRO AM, KCPI=>VGID, KCRN=REMOTEQ
FPUT NE, KCRN=>VGID
...
PEND FI
```

6 Programmaufbau bei Kommunikation mit Transportsystem-Anwendungen

Dieses Kapitel beschreibt die Besonderheiten, die Sie beachten müssen, wenn Sie eine Kommunikation mit Transportsystem-Anwendungen (= TS-Anwendungen) programmieren möchten.

Überblick

Eine Transportsystem-Anwendung wird als Client mit Partnertyp APPLI oder SOCKET an eine UTM-Anwendung angeschlossen. Bei Partnertyp SOCKET muss der Transportsystem-Endpunkt mit T-PROT=(SOCKET[, *ANY|*USP [, SECURE]]) generiert sein und die Kommunikation oberhalb von TCP/IP muss über das **U**TM-**S**ocket-**P**rotokoll (USP) erfolgen. Sie können die Transport-Anwendung wie folgt generieren:

- Explizit mit einer PTERM-/LTERM-Anweisung:

  ```
  PTERM ...,PTYPE=APPLI/SOCKET[, BCAMAPPL= socket-anwendungsname ]
  LTERM ...
  ```

- Als LTERM-Pool mit einer TPOOL-Anweisung:

  ```
  TPOOL ...,PTYPE=APPLI/SOCKET[, BCAMAPPL= socket-anwendungsname ]
  ```

Für TS-Anwendungen vom Type SOCKET müssen Sie einen zusätzlichen Anwendungsnamen generieren, der als Transportsystem-Endpunkt verwendet wird:

```
BCAMAPPL socket-anwendungsname , T-PROT=(SOCKET[, *ANY|*USP[, SECURE]])
```

Von einer Transportsystem-Anwendung aus kann in der UTM-Anwendung

- ein Vorgang gestartet werden
- oder eine Nachricht für eine TAC-Queue oder eine Temporäre Queue erzeugt werden.

Informationen zur Generierung von Transportsystem-Anwendungen finden Sie im openUTM-Handbuch „Anwendungen generieren“.

6.1 Kommunikation mit TS-Anwendungen vom Typ APPLI

Für eine Eingabenachricht von der TS-Anwendung vom Typ APPLI gilt:

- Soll mit einer an die UTM-Anwendung gerichteten Nachricht ein Vorgang gestartet werden, dann muss der TAC am Beginn der Nachricht stehen.
- Soll die Nachricht in die Message Queue einer TAC-Queue oder einer Temporären Queue eingehängt werden, so muss der Name der Queue am Beginn der Nachricht stehen. Geht die Nachricht an eine Temporäre Queue, dann darf unter dem Namen der Temporären Queue darf kein TAC oder LTERM in der Anwendung existieren.
- Ist der TAC kürzer als acht Zeichen, muss er durch mindestens ein Leerzeichen vom Rest der Nachricht getrennt sein.
- Ist der angegebene TAC oder der Name der Queue ungültig, wird der Event-Service BADTACS gestartet, sofern er generiert ist. Andernfalls wird die Meldung K009 an die TS-Anwendung gesendet, außer es wird ein eigener Meldungsmodul verwendet, in dem für die Meldung K009 nicht PARTNER als Meldungsziel generiert wurde.

Wenn die TS-Anwendung einen Vorgang startet, kann dieser die Nachricht mit dem KDCS-Aufruf MGET oder FGET lesen. Wenn die TS-Anwendung eine Nachricht an eine Service-gesteuerte Queue sendet, kann ein beliebiger Vorgang die Nachricht mit dem KDCS-Aufruf DGET lesen. Ist beim Lese-Aufruf die Länge KCLA kleiner als die Länge der Teilnachricht, wird nur der angeforderte Teil gelesen; der Rest der Nachricht geht verloren. Der Returncode 01Z zeigt an, dass die Nachricht nicht vollständig gelesen wurde.

Bitte beachten Sie, dass die TS-Anwendung den Namen des TAC oder der Queue in dem für die UTM-Anwendung "richtigen" Code bereitstellen muss, d.h. in EBCDIC für eine UTM-Anwendung auf BS2000-Systemen und in ASCII für eine UTM-Anwendung auf Unix-, Linux- oder Windows-Systemen.

Alternativ kann in Anwendungen auf Unix-, Linux- oder Windows-Systemen für die Verbindung zur TS-Anwendung Code-Konvertierung generiert werden (Operand MAP=SYS1/SYS2/SYS3/SYS4 in der PTERM- oder TPOOL-Anweisung).

6.2 Kommunikation mit Socket-USP-Anwendungen

openUTM arbeitet nachrichtenorientiert und startet ein Teilprogramm erst, nachdem eine vollständige Nachricht für das Teilprogramm empfangen wurde. Die Socket-Schnittstelle dagegen ist eine Bytestream-Schnittstelle.
Um innerhalb der Bytestreams Nachrichtengrenzen zur erkennen, benötigt openUTM oberhalb von TCP/IP ein zusätzliches Protokoll. Diesem Zweck dient das von openUTM zur Verfügung gestellte **U**TM-**S**ocket-**P**rotokoll (**USP**), das die Umsetzung der über die Socket-Schnittstelle empfangenen Bytestreams in Nachrichten ermöglicht (s. "Aufbau des Socket-Protokoll-Headers"). Als Transportprotokoll wird TCP/IP vorausgesetzt.

Beispielprogramme

Mit openUTM werden die Socket-Client-Beispielprogramme SOCBSP und USOCV6 sowie ein Teilprogramm SOCMIRR für Socket-Partner zur Verfügung gestellt. SOCMIRR ("socket mirror") sendet die empfangenen Nachrichten an den Socket-Client zurück.

Auf BS2000-Systemen finden Sie die Sourcen und die Objektmodule der Beispielprogramme in der Bibliothek SYSLIB.UTM.070.EXAMPLE.

Auf Unix- und Linux-Systemen sind SOCBSP (Source `socbsp.c`), USOCV6 (Source `usocv6.c`) und SOCMIRR (Source `socmirr.c`) Bestandteil der Beispiel-Anwendung. Die und Sourcen finden Sie unter *utmsample*/`soc-c` bzw. *utmsample*/`utm-c` (*utmsample* = Verzeichnis der Beispielanwendung).

Auf Windows-Systemen sind SOCBSP (Source `socbsp.c`), USOCV6 (Source `usocv6.c`) und SOCMIRR (Source `socmirr.c`) Bestandteil des QuickStartKit . Die Sourcen finden Sie im Verzeichnis *utmpfad*`\utmsample\soc-c` bzw. *utmpfad*`\utmsample\utm-c`.

6.2.1 Eingabe-Nachrichten für openUTM

Bei Eingabe-Nachrichten von Socket-Anwendungen muss ein entsprechender Protokoll-Header in der Partner-Anwendung aufgebaut und jeder Teilnachricht bzw. jedem Fragment vorangestellt werden. Dieser Protokoll-Header wird abgeschnitten und nicht an das Teilprogramm übergeben.

Für die dem Protokoll-Header folgenden Daten gilt:

- Soll mit einer an openUTM gerichteten Nachricht ein Vorgang gestartet werden, dann muss der TAC am Beginn der Nachricht stehen.
- Soll die Nachricht in die Message Queue einer TAC-Queue oder einer Temporären Queue eingehängt werden, so muss der Name der Queue am Beginn der Nachricht stehen. Geht die Nachricht an eine Temporäre Queue, dann darf unter dem Namen der Temporären Queue darf kein TAC oder LTERM in der Anwendung existieren.
- Ist der TAC kürzer als acht Zeichen, muss er durch mindestens ein Leerzeichen vom Rest der Nachricht getrennt sein.
- Ist der angegebene TAC oder der Name der Queue ungültig, wird der Event-Service BADTACS gestartet, sofern er generiert ist. Andernfalls wird die Meldung K009 an die Socket-Anwendung gesendet, außer es wird ein eigener Meldungsmodul verwendet, in dem für die Meldung K009 nicht PARTNER als Meldungsziel generiert wurde.

Wenn die TS-Anwendung einen Vorgang startet, kann dieser die (Teil-)Nachricht(en) mit dem KDCS-Aufruf MGET oder FGET gelesen. Wenn die TS-Anwendung eine oder mehrere Nachricht(en) an eine Service-gesteuerte Queue sendet, kann ein beliebiger Vorgang die (Teil-)Nachricht(en) mit dem KDCS-Aufruf DGET lesen. Dabei gilt:

- Ist beim MGET-Aufruf die Länge KCLA kleiner als die Länge der Teilnachricht, wird nur der angeforderte Teil gelesen; der Rest der Nachricht geht jedoch nicht verloren. Der Returncode 02Z zeigt an, dass die Teilnachricht nicht vollständig gelesen wurde. Mit dem nächsten Lese-Aufruf kann der Rest der Teilnachricht gelesen werden, mit dem übernächsten eine weitere Teilnachricht usw.
- Ist beim DGET-/FGET-Aufruf die Länge KCLA kleiner als die Länge der Teilnachricht, wird nur der angeforderte Teil gelesen; der Rest der Nachricht geht verloren. Der Returncode 01Z zeigt an, dass die Teilnachricht nicht vollständig gelesen wurde. Mit dem nächsten Lese-Aufruf wird eine weitere Teilnachricht gelesen.

Sind alle Teilnachrichten gelesen, wird dies durch den Returncode 10Z angezeigt.

Wenn für die Verbindung zum Socket-Partner keine Code-Konvertierung generiert wurde (PTERM- oder TPOOL-Anweisung, Operand MAP=USER), muss die Socket-Anwendung den Namen des oder der Queue in dem für die UTM-Anwendung "richtigen" Code bereitstellen, d.h. in EBCDIC für eine UTM(BS2000)-Anwendung und in ASCII für eine UTM-Anwendung auf Unix-, Linux- oder Windows-Systemen.

6.2.2 Ausgabe-Nachrichten von openUTM

Beim Senden von Nachrichten an Socket-Partner muss jede Teilnachricht bzw. jedes Fragment mit einem eigenen MPUT NT/NE-Aufruf verschickt werden.
Bei jedem MPUT-Aufruf wird eine eigene Teilnachricht erzeugt, auch wenn die Länge Null angegeben wird. Eine Nachricht mit der Länge 0 wird jedoch nur gesendet, wenn für die zu sendende Nachricht automatisch ein USP-Header erzeugt und der Nachricht vorangestellt wird (vgl. Tabelle unten). Ausnahme: Wenn das Teilprogramm nur einen einzigen MPUT-Aufruf enthält, wird unabhängig vom Wert des Parameters USP-HDR keine Nachricht gesendet.

Wenn der Socket-Partner beim Nachrichtenempfang einen USP-Header erwartet, können Sie per Generierung festlegen, dass openUTM bei Nachrichten oder Meldungen an den Socket-Partner automatisch einen USP-Header erzeugt und der Nachricht voranstellt.

Dies wird bei der Generierung durch den Operanden USP-HDR= in der PTERM- oder TPOOL-Anweisung festgelegt:

- Für alle Nachrichten, d.h. K-Meldungen + MPUT/FPUT-Nachrichten (USP-HDR=ALL)
- Nur für K-Meldungen (USP-HDR=MSG)
- Keinen Header (USP-HDR = NO)

Wenn für den Socket-Partner keine Code-Konvertierung generiert wurde (Operand MAP=USER in der PTERM- oder TPOOL-Anweisung), können Sie den USP-Header auch im Teilprogramm selbst erzeugen und der Nachricht voranstellen.

Informationen zur Generierung von Socket-Anwendungen finden Sie im openUTM-Handbuch „Anwendungen generieren“.

Austausch von langen Nachrichten

Mit Socket-Partnern können Eingabe-Nachrichten und Dialog-Ausgabe-Nachrichten beliebiger Länge ausgetauscht werden. Falls eine Gesamtnachricht länger ist als 32767 Byte (Ausgabe) bzw. 32000 Bytes (Eingabe), dann muss die Nachricht **fragmentiert** werden, d.h. sie muss aus mehreren Teilnachrichten bestehen. Dabei darf jede Teilnachricht bei der Ausgabe maximal 32767 Byte (inclusive USP-Header) und bei der Eingabe maximal 32000 Byte lang sein.

Teilnachrichten sind dadurch gekennzeichnet, dass im USP-Header das entsprechende Flagfeld und der entsprechenden Messagetyp gesetzt sind, siehe unten. Bei Ausgaben muss das Programm diese Werte setzen.

Fragmentierte Eingabe-Nachrichten müssen mit einer entsprechenden Anzahl von MGET/FGET/DGET NT Aufrufen gelesen werden. Fragmentierte Dialog-Ausgabe-Nachrichten müssen mit einer entsprechenden Anzahl von MPUT NT Aufrufen gesendet werden.

Asynchron-Ausgabe-Nachrichten (FPUT/DPUT) können nicht fragmentiert werden. Sie dürfen nicht länger als 32700 Bytes lang sein.

6.2.3 Aufbau des USP-Headers

Der Header enthält den Identifier, zwei Versionsfelder und ein Flagfeld, ein Typfeld und ein Längenfeld. Das beschriebene Protokoll wird von openUTM eingabeseitig erwartet. Es ist wie folgt aufgebaut:

Identifier ASCII, 4 Byte	**Version Major 1 Byte**	**Version Minor 1 Byte**	**Flags 1 Byte**	**MsgType 1 Byte**	**MsgSize 4 Byte**	**Daten max 31988 Byte**
| UTMS(55544D53)	| 01	| 01		| Typ der Nachricht	| 0000xxxx	| Daten

Erläuterung

Identifier

Das Identifikationsfeld muss für openUTM immer den ASCII-String "UTMS" (=0x55544D53) enthalten

Version Major

Im Versionsfeld Major muss immer 0x01 stehen.

Version Minor

Im Versionsfeld Minor muss der Wert 0x01 stehen.

Flags

Das Flagfeld ist ein Informationsfeld. Nur Bit `0x02` wird ausgewertet: Ist es gesetzt, folgt der Nachricht ein weiteres Fragment. Ist es nicht gesetzt, folgt kein Fragment. Alle anderen Bits sind für künftige Versionen reserviert.

MsgType

Im Typfeld kann 0x00, 0x01 oder 0x07 stehen:

- Nachrichten vom Client empfangen:
  Bei unfragmentierten Nachrichten setzt der Client `MsgType` auf den Wert 0x00. Bei fragmentierten Nachrichten setzt der Client `MsgType` beim ersten Fragment auf den Wert 0x00 (Bit 0x02 im Flagfeld ist gesetzt). Bei den Folgefragmenten wird `MsgType` auf den Wert 0x07 gesetzt (Bit 0x02 bleibt gesetzt). Beim letzten Folgefragment ist Bit 0x02 nicht gesetzt.
- Nachrichten an Client senden:
  Wird ein USP-Header erzeugt (z.B. bei USP-HDR=ALL), dann setzt openUTM das Feld `MsgType` auf den Wert 0x01, bei Folgefragmenten auf 0x07.

MsgSize

Das Längenfeld `MsgSize` enthält die Länge der (Teil)nachricht einschließlich Header. Diese Länge darf eingabeseitig 32000 Byte und ausgabeseitig 32767 Byte nicht überschreiten. Die Nachrichtenlänge wird in Network Byte Order (Big Endian) übertragen. Mit Hilfe der C-Funktionen `htonl` (Host to network long) und `ntohl` (Network to Host long) kann zwischen lokaler Darstellung in Netzwerksicht konvertiert werden.

Beispiele

1. Der Client sendet Nachrichten an openUTM, siehe mit ausgelieferte Beispielsource SOCBSP.C:
   - Der Client sendet eine Nachricht (ohne Fragmentierung)
   - Der Client sendet 2 Nachrichtenteile
   - Der Client sendet 3 Nachrichtenteile

Anzahl Nachrichtenteile	Identifier	Major Version	Minor Version	Flag	MsgType	Size
1 Gesamtnachricht	55544D53	01	01	00	00	0000nnnn
2 Teile						
○ Teil 1	55544D53	01	01	02	00	0000nnnn
○ Teil 2	55544D53	01	01	00	07	0000mmmm
3 Teile						
○ Teil 1	55544D53	01	01	02	00	0000kkkk
○ Teil 2	55544D53	01	01	02	07	0000nnnn
○ Teil 3	55544D53	01	01	00	07	0000mmmm

2. Der Server sendet Nachrichten an den Client (es wird ein USP-Header erzeugt):
   - Eine Gesamtnachricht (ohne Fragmentierung)
   - 3 Nachrichtenteile

Anzahl Nachrichtenteile	Identifier	Major Version	Minor Version	Flag	MsgType	Size
1 Gesamtnachricht	55544D53	01	01	00	01	0000nnnn
3 Teile						
○ Teil 1	55544D53	01	01	02	01	0000kkkk
○ Teil 2	55544D53	01	01	02	07	0000nnnn
○ Teil 3		01	01	00	07	0000mmmm

kkkk, nnnn, *mmmm* sind die jeweiligen Längen der einzelnen Nachrichtenteile.

7 KDCS-Aufrufe

In diesem Kapitel finden Sie alle Informationen, die Sie zum Einsatz der Programmschnittstelle KDCS in Ihrem Programm benötigen. Mit dem KDCS-Aufruf rufen Sie openUTM auf. Anhand der Angaben im KDCS-Parameterbereich erkennt openUTM die gewünschte Funktion und führt sie aus.

7.1 Übersicht über alle KDCS-Aufrufe

Die UTM-Aufrufe realisieren die Schnittstelle KDCS, die in DIN 66 265 genormt ist ("Schnittstellen eines Kerns für transaktionsorientierte Anwendungssysteme"). Die UTM-Programmschnittstelle ist eine aufwärtskompatible Erweiterung dieser DIN-Norm. Die folgende Tabelle zeigt diese Erweiterungen.

Aufruf	Bestandteil von DIN 66 265	Erweiterungen gegenüber DIN 66 265
APRO	nein	verteilte Verarbeitung
CTRL	nein	verteilte Verarbeitung (nur für OSI TP)
DADM	nein	Administration von Message Queues (Asynchron-Aufträgen)
DGET	nein	aus Service-gesteuerten Message Queues lesen
DPUT	ja	Quittungsaufträge und Benutzerinformationen
FGET	ja	verteilte Verarbeitung
FPUT	ja	verteilte Verarbeitung
GTDA	ja	keine
INFO	nein	Informationsdienste
INIT	ja	verteilte Verarbeitung
LPUT	ja	keine
MCOM	nein	Definition von Auftrags-Komplexen
MGET	ja	verteilte Verarbeitung
MPUT	ja	verteilte Verarbeitung
PADM	nein	Steuern von Druckern und Druckausgaben
PEND	ja	verteilte Verarbeitung, PEND KP/PS/FC/SP/RS/FR nicht in DIN 66 265 enthalten
PGWT	nein	Programmverwaltung
PTDA	ja	keine
QCRE	nein	Temporäre Queue erzeugen
QREL	nein	Temporäre Queue löschen
RSET	nein	Rücksetzoperation
SGET	ja	keine
SIGN	nein	An- und Abmelden, Passwortänderung, Berechtigungsdaten überprüfen

Aufruf	Bestandteil von DIN 66 265	Erweiterungen gegenüber DIN 66 265
SPUT	ja	keine
SREL	ja	keine
UNLK	nein	Entsperren von Speicherbereichen

Welche KDCS-Aufrufe es gibt und welche Funktion sie haben, entnehmen Sie der folgenden Tabelle:

Aufruf	Funktion	Funktionsgruppe
APRO	Adressieren eines Auftragnehmer-Vorgangs	Nachrichtenkommunikation (bei verteilter Verarbeitung)
CTRL	Steuern eines OSI TP-Dialogs	Nachrichtenkommunikation (bei verteilter Verarbeitung)
DADM	Administration von Asynchron-Aufträgen	Verwaltung von Message Queues und Druckern
DGET	Lesen von Nachrichten aus einer Service-gesteuerten Message Queue	Nachrichtenkommunikation - Message Queuing
DPUT	Erzeugen von zeitgesteuerten Asynchron-Aufträgen und von Quittungsaufträgen	Nachrichtenkommunikation - Message Queuing
FGET	Empfangen von Asynchron-Nachrichten	Nachrichtenkommunikation - Message Queuing
FPUT	Erzeugen von Asynchron-Aufträgen	Nachrichtenkommunikation - Message Queuing
GTDA	Lesen aus einem TLS	Speicherverwaltung
INFO	Informationen abrufen	Informationsdienste
INIT	Anmelden eines Programms bei openUTM	Programmverwaltung
LPUT	Schreiben in die Protokolldatei	Protokollierung
MCOM	Definieren eines Auftrags-Komplexes	Nachrichtenkommunikation
MGET	Empfangen einer Dialog-Nachricht	Nachrichtenkommunikation - Dialog
MPUT	Senden einer Dialog-Nachricht	Nachrichtenkommunikation - Dialog
PADM	Steuern von Druckern und Druckausgaben	Verwaltung von Message Queues und Druckern
PEND	Beenden des Programms	Programmverwaltung

Aufruf	Funktion	Funktionsgruppe
PGWT	Wartepunkt in einem Teilprogrammlauf setzen	Programmverwaltung
PTDA	Schreiben in einen TLS	Speicherverwaltung
QCRE	Erzeugen von Temporären Message Queues	Verwaltung von Temporären Message Queues
QREL	Löschen von Temporären Message Queues	Verwaltung von Temporären Message Queues
RSET	Rücksetzen angeforderter Änderungen und Operationen	Programmverwaltung
SGET	Lesen aus einem Sekundärspeicherbereich	Speicherverwaltung
SIGN	An- und Abmelden, Passwortänderung, Berechtigungsdaten überprüfen	Anmeldeverwaltung
SPUT	Schreiben in einen Sekundärspeicherbereich	Speicherverwaltung
SREL	Freigeben eines Sekundärspeicherbereichs	Speicherverwaltung
UNLK	Entsperren eines TLS, ULS oder GSSB	Speicherverwaltung

Die nächste Tabelle zeigt, welche Aufrufe bei welcher Funktionsgruppe eine Rolle spielen.

Funktionsgruppe	Zugehörige UTM-Aufrufe
Anmeldeverwaltung	SIGN
Programmverwaltung	INIT, RSET, PEND, PGWT
Nachrichtenkommunikation - Dialog	MGET, MPUT
Nachrichtenkommunikation - Message Queuing	MCOM, DGET, DPUT, FGET, FPUT
Verwaltung von Message Queues und Druckern	DADM, PADM
Verwaltung von Temporären Message Queues	QCRE, QREL
Speicherverwaltung	GTDA, PTDA, SGET, SPUT, SREL, UNLK
Informationsdienste	INFO
Protokollierung	LPUT
verteilte Verarbeitung	APRO, CTRL, DPUT, FPUT, INIT, MCOM, MGET, MPUT, PEND, RSET, PGWT

openUTM gibt nach jedem Aufruf (außer PEND) Informationen im KDCS-Kommunikationsbereich zurück.

Die Parameterfelder, die Sie vor dem Aufruf versorgen bzw. nach Ausführung zurückerhalten, haben bestimmte Namen. Dies soll Ihnen helfen, mit der Schnittstelle KDCS und ihrer Beschreibung besser zurechtzukommen. Ihre Verwendung soll auch die Wartbarkeit und Übertragbarkeit der Programme erhöhen. Im Anhang werden alle Operandenfelder und ihre Verwendung bei den verschiedenen Aufrufen im Überblick gezeigt.

7.2 Hinweise zur Beschreibung der KDCS-Aufrufe

Die KDCS-Aufrufe sind in diesem Kapitel in alphabetischer Reihenfolge beschrieben. Dies erleichtert Ihnen das Nachschlagen.

Jede Beschreibung eines Aufrufs besteht aus vier Teilen:

- Zunächst finden Sie eine Beschreibung der verschiedenen Funktionen eines Aufrufs.
- Danach folgt eine tabellarische Darstellung des Aufrufs mit allen notwendigen Angaben. Die Felder, die Sie vor dem Aufruf versorgen müssen, sind grau unterlegt:

grau	Dieses Feld müssen Sie vor dem Aufruf versorgen

  Dabei werden zu jedem Aufruf auch die entsprechenden C/C++-Makros aufgeführt. Eine ausführliche Beschreibung dieser Makros finden Sie in Abschnitt „C/C++-Makroschnittstelle“.
- Dieser Darstellung folgt jeweils die Beschreibung der Angaben im KDCS-Parameterbereich und im 2. Parameter sowie der Rückgaben von openUTM.
- Zuletzt wird auf die besonderen Eigenschaften des Aufrufs eingegangen.

Ist in einer Tabelle "—" eingetragen, dann ist der Eintrag für die betreffende Funktion irrelevant.

> ¡ Die Feldnamen der KDCS-Schnittstelle sind für COBOL und C/C++ weitgehend identisch und unterscheiden sich nur bezüglich Groß-/Kleinschreibung. Überall dort, wo darüber hinausgehende Unterschiede bestehen, ist der Feldname für C/C++ jeweils hinter dem COBOL-Feldnamen angegeben (durch Schrägstrich getrennt), z.B.: "KCTAG/kcday".

> ¡ Der Returncode 79Z ist ein allgemeiner Fehlercode der anzeigt, dass der Wert im Feld KCOP=Operationscode einen ungültigen Wert hat. Er kann deshalb keinem Operationscode zugeordnet werden.

7.3 APRO Adressieren eines Auftragnehmer-Vorgangs

Mit dem Aufruf APRO (address program) adressieren Sie im Auftraggeber-Vorgang einen Auftragnehmer-Vorgang oder eine TAC-Queue. Der APRO-Aufruf ist nur bei verteilter Verarbeitung anwendbar. Die Daten an den Auftragnehmer-Vorgang werden mit MPUT bzw. FPUT/DPUT gesendet, wobei man bei diesen Aufrufen als Empfänger die im APRO-Aufruf definierte Vorgangs-Id angibt.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich						
	KCOP	KCOM	KCLM	KCRN	KCPA	KCPI	KCOF
Adressieren eines Dialog-Vorgangs	"APRO"	"DM"	0/19/58	LTAC-Name	(OSI-)LPAP-Name/Master-LPAP-Name /Leerzeichen	Vorgangs-Id	erlaubte OSI-Funktion
Adressieren eines Asynchron-Vorgangs oder einer TAC-Queue	"APRO"	"AM"	0/19/58	LTAC-Name	(OSI-)LPAP-Name/Master-LPAP-Name /Leerzeichen	Vorgangs-Id	erlaubte OSI-Funktion

Die Angabe im Feld KCPA ist abhängig von der Art der Adressierung:

- bei einstufiger Adressierung muss das Feld Leerzeichen enthalten,
- bei zweistufiger Adressierung muss das Feld den Namen der Partner-Anwendung enthalten ((OSI-)LPAP-Name bzw. Master-LPAP-Name).

Näheres zur ein- bzw. zweistufigen Adressierung von Auftragnehmer-Vorgängen bei verteilter Verarbeitung finden Sie im openUTM-Handbuch „Konzepte und Funktionen“.

Versorgen des 2. Parameters (Auswahl spezieller OSI TP-Funktionskombinationen)

Der 2. Parameterbereich findet nur Verwendung bei der Kommunikation über das OSI TP-Protokoll. Er erlaubt es, andere Funktionskombinationen zu spezifizieren, als über die Standardauswahl mittels des KDCS-Parameters KCOF möglich sind. Außerdem kann ausgewählt werden, ob SIGNON-Daten an den Auftragnehmer-Vorgang übergeben werden sollen. Für den 2. Parameterbereich ist eine Sprach-spezifische Datenstruktur verfügbar; für COBOL im COPY-Element KCAPROC, für C/C++ in der Include-Datei *kcapro.h*.

Wenn der 2. Parameterbereich verwendet wird, dann muss im Feld KCLM der Wert 19 bzw. 58 als Länge der Datenstruktur angegeben werden, und das Feld KCOF muss den Wert "O" enthalten.

Versorgen der Parameter im KDCS-Parameterbereich	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"APRO"
KCOM	"DM"/"AM"
KCLM	0/19/58
KCRN	LTAC-Name
KCPA	(OSI-)LPAP-Name/Master-LPAP-Name/Leerzeichen
KCPI	Vorgangs-Id
KCOF	OSI-Funktionen

Versorgen der Parameter im 2. Parameterbereich (nur bei KCOF=O)	
Feldname im 2. Parameterbereich	Inhalt
KCVERS	Versionsnummer der Datenstruktur
KCFUPOL	Polarized FU (Y)
KCFUHSH	Handshake FU (Y/N)
KCFUCOM	Commit FU (Y/N)
KCFUCHN	Chained FU (Y/Leerzeichen)
KCFUFILL	leer - für spätere Erweiterungen
KCSECTYP	Security-Typ (N/S/P)
KCUIDTYP	Datentyp der Benutzerkennung (P/T/O)
KCUIDLTH	Länge der Benutzerkennung
KCUSERID	Benutzerkennung
KCSECFIL	leer - für spätere Erweiterungen
KCPWDTYP	Datentyp des Passworts (P/T/O)
KCPWDLTH	Länge des Passworts
KCPSWORD	Passwort

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	2. Parameterbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_APRODM / KDCS_APROAM	(kcrn,kcpa,kcpi)
KDCS_APRODM_OSI / KDCS_APROAM_OSI	(kcrn,kcpa,kcpi,kcof)
KDCS_APRODM_OSI_O / KDCS_APROAM_OSI_O	(nb,kclm,kcrn,kcpa,kcpi)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich machen Sie für den APRO-Aufruf folgende Angaben:

KCOP

Im Feld KCOP geben Sie den Operationscode APRO an.

KCOM

Im Feld KCOM tragen Sie die Operationsmodifikation ein:

- DM (Dialog Message) zur Adressierung eines Dialog-Vorgangs
- AM (Asynchronous Message) zur Adressierung eines Asynchron-Vorgangs oder einer TAC-Queue.

KCLM

Im Feld KCLM geben Sie die Länge des 2. Parameterbereichs an:

- falls kein 2. Parameterbereich, verwendet wird, die Länge 0.
- falls ein 2. Parameterbereich verwendet wird und Security-Typ "N" oder "S" gewählt ist, die Länge 19.
- falls ein 2. Parameterbereich verwendet wird und Security-Typ "P" gewählt ist, die Länge 58.

KCRN

Im Feld KCRN geben Sie den logischen Transaktionscode (LTAC-Namen) des Auftragnehmer-Vorgangs an.

KCPA

Ob Sie im Feld KCPA die Partner-Anwendung identifizieren müssen, hängt von der Art der Adressierung ab:

- Bei zweistufiger Adressierung tragen Sie den logischen Namen der Partner-Anwendung ein, d.h. den LPAP-Namen, OSI-LPAP-Namen oder Master-LPAP-Namen eines OSI-LPAP- oder LU6.1-LPAP-Bündels.
- Bei einstufiger Adressierung tragen Sie Leerzeichen ein (der Name der Partner-Anwendung wird dann aus der Generierungsanweisung LTAC genommen).

Die Angabe der Partner-Anwendung in KCPA hat Priorität vor der bei der Generierung in der LTAC-Anweisung spezifizierten Partner-Anwendung.

KCPI

Im Feld KCPI vergeben Sie die Vorgangs-Id, mit der der Auftragnehmer-Vorgang im Auftraggeber-Vorgang bei den Aufrufen MPUT, MCOM, FPUT, DPUT bzw. MGET angesprochen werden soll. Die Vorgangs-Id muss mit dem Zeichen ">" beginnen.

KCOF

Das Feld KCOF enthält die bei der Kommunikation mit einem OSI TP-Partner erlaubten Funktionen (bei Kommunikation über das LU6.1-Protokoll irrelevant).
Mögliche Werte:

- **B** (basic functions)
  Basisfunktionen
- **H** (handshake functions)
  Basis- und Handshake-Funktionen (nur bei APRO DM möglich)
- **C** (chained transactions)
  Basis- und Commit-Funktionen mit Chained Transactions
- **O** (other combination)
  Die Auswahl der Funktionen geschieht über den 2. Parameterbereich; d.h. wenn KCOF=O angegeben wird, dann muss bei dem KDCS-Aufruf ein 2. Parameterbereich übergeben werden.

> ¡ Bei der Adressierung eines OSI TP-Auftragnehmers müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Im 2. Parameterbereich geben Sie an:

KCVERS

Im Feld KCVERS geben Sie die Versionsnummer der Datenstruktur an: in dieser openUTM-Version ist dies 1.

KCFUPOL

in das Feld KCFUPOL tragen Sie ein, ob die Functional Unit "Polarized Control" ausgewählt werden soll. Da "Shared Control" in dieser Version nicht unterstützt wird, ist hier nur die Angabe "Y" zulässig.

KCFUHSH

In das Feld KCFUHSH tragen Sie ein, ob die Functional Unit "Handshake" ausgewählt werden soll (Y/N). Wird ein Asynchron-Vorgang adressiert (APRO AM), muss für KCFUHSH "N" angegeben werden.

KCFUCOM

In das Feld KCFUCOM tragen Sie ein, ob die Functional Unit "Commit" ausgewählt werden soll (Y/N). Die Functional Unit "Commit" kann nur ausgewählt werden, wenn der adressierte OSI-LPAP-Partner im Application Context die abstrakte Syntax CCR enthält (**C**ommitment, **C**oncurrency and **R**ecovery).

KCFUCHN

In das Feld KCFUCHN tragen Sie ein, ob die Functional Unit "Chained Transactions" ausgewählt werden soll. Dieses Feld ist nur von Bedeutung, wenn auch die Functional Unit "Commit" ausgewählt wurde, d.h. KCFUCOM=Y angegeben ist.
Da "Unchained Transactions" in dieser Version nicht unterstützt wird, ist hier - falls Functional Unit "Commit" ausgewählt wurde - nur die Angabe "Y" zulässig.
Falls die Functional Unit "Commit" **nicht** ausgewählt wurde, ist das Feld KCFUCHN bedeutungslos. In diesem Fall müssen Sie ein Leerzeichen eintragen.

KCSECTYP

Im Feld KCSECTYP wählen Sie den Security-Typ aus.
Der Security-Typ regelt, ob SIGNON-Daten (Benutzerkennung und Passwort) an den Auftragnehmer-Vorgang übergeben werden:

- N (none)
  Es werden keine SIGNON-Daten an den Auftragnehmer-Vorgang übergeben.
- S (same)
  Es wird die Benutzerkennung an den Auftragnehmer-Vorgang übergeben, unter der der lokale Vorgang läuft.
- P (program)
  Es werden die in den Feldern KCUSERID und KCPSWORD spezifizierten Werte als Benutzerkennung und Passwort an den Auftragnehmer-Vorgang übergeben.

Security-Typ "S" oder "P" können Sie nur dann auswählen, wenn der adressierte OSI-LPAP-Partner im Application Context die abstrakte Syntax UTMSEC enthält.

KCUIDTYP

Im Feld KCUIDTYP tragen Sie den Datentyp der im Feld KCUSERID angegebenen Benutzerkennung ein:

- P: Der im Feld KCUSERID angegebene String ist vom Typ "Printable String".
- T: Der im Feld KCUSERID angegebene String ist vom Typ "T61 String".
- O: Der im Feld KCUSERID angegebene String ist vom Typ "Octet String".Octet String ist eine hexadezimale Zeichenfolge. Es erfolgt keine Code-Umsetzung.

Die Wertebereiche dieser Datentypen sind im openUTM-Handbuch „Anwendungen generieren“ bei der KDCDEF-Anweisung LTAC beschrieben.

KCUIDLTH

Im Feld **KCUIDLTH** tragen Sie die Länge der im Feld KCUSERID angegebenen Benutzerkennung ein (in Byte, maximal 16).

KCUSERID

Im Feld **KCUSERID** geben Sie die Benutzerkennung an, die - falls KCSECTYP=P gesetzt ist - an den Auftragnehmer-Vorgang übergeben wird.

KCPWDTYP

Im Feld **KCPWDTYP** tragen Sie den Datentyp des im Feld KCPSWORD angegebenen Passworts ein:

- P: Der im Feld KCPSWORD angegebene String ist vom Typ "Printable String".
- T: Der im Feld KCPSWORD angegebene String ist vom Typ "T61 String".
- O: Der im Feld KCPSWORD angegebene String ist vom Typ "Octet String" (siehe auch Feld KCUIDTYP).

Die Wertebereiche dieser Datentypen sind im openUTM-Handbuch „Anwendungen generieren“ bei der KDCDEF-Anweisung LTAC beschrieben.

KCPWDLTH

Im Feld **KCPWDLTH** tragen Sie die Länge des im Feld KCPSWORD angegebenen Passworts ein (in Byte, maximal 16). Falls kein Passwort übergeben wird, muss null angegeben werden.

KCPSWORD

Im Feld **KCPSWORD** geben Sie das Passwort an, das - falls KCSECTYP=P gesetzt ist - an den Auftragnehmer-Vorgang übergeben wird.

> ¡ Nicht verwendete Felder des 2. Parameterbereichs müssen mit Leerzeichen versorgt werden. Ausnahme: Nicht verwendete Längenfelder müssen mit null versorgt werden.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

Die Adresse des KDCS-Parameterbereichs.

2. Parameter

(wenn benötigt): Die Adresse des 2. Parameterbereichs (Auswahl spezieller OSI TP-Funktionskombinationen).

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe unten.

KCRCDC

KDCS-Returncodes im Feld KCRCCC beim APRO-Aufruf

Im Programm sind auswertbar:

000	Die Funktion wurde ausgeführt.
40Z	openUTM kann die Funktion nicht durchführen. Es liegt ein Generierungs- oder ein Systemfehler vor, oder die Funktion APRO wurde aufgerufen und die Anwendung ohne verteilte Verarbeitung generiert, oder es ist zurzeit keine Verbindung zur Partner-Anwendung möglich (siehe Eintrag in KCRCDC).

41Z Unzulässiger APRO-Aufruf. Dies kann z.B. einen der folgenden Gründe haben:

- Der APRO-Aufruf wurde in einem Anmelde-Vorgang angegeben.
- In einem Vorgang soll mit einigen Partnern über LU6.1 kommuniziert werden und mit anderen über das OSI TP-Protokoll mit Functional Unit "Commit".
- In einer verteilten Transaktion über das OSI TP-Protokoll sollen Associations über mehr als einen lokalen ACCESS-POINT aufgebaut werden.
- Eine Association mit Functional Unit "Commit" soll aufgebaut werden. Die abstrakte Syntax CCR (**C**ommitment, **C**oncurrency and **R**ecovery) ist jedoch **nicht** Im Application Context des zugehörigen OSI-LPAP-Partners enthalten.
- Eine Association mit Security-Typ "S" oder "P" soll aufgebaut werden. Im Application Context des zugehörigen OSI-LPAP-Partners ist jedoch die abstrakte Syntax UTMSEC **nicht** enthalten.

42Z Der Eintrag in KCOM ist ungültig.

43Z Der Eintrag in KCLM ist ungültig.

44Z Der Wert in KCRN bezeichnet keinen gültigen LTAC eines Auftragnehmer-Vorgangs.
Mögliche Gründe:

- Entsprechender LTAC ist in der Konfiguration nicht bekannt.
- LTAC ist gesperrt.
- Benutzerkennung hat keinen Keycode für den LTAC.
- Eintrag in KCOM passt nicht zum angegebenen LTAC (KCOM mit Wert DM und LTAC eines Asynchron-Vorgangs oder einer TAC-Queue; oder KCOM mit Wert AM und LTAC eines Dialog-Vorgangs).

46Z Der Eintrag in KCPA ist ungültig, d.h. unter dem angegebenen Namen ist keine Partner-Anwendung generiert oder es wurden in KCPA Leerzeichen angegeben und in der LTAC-Generierungsanweisung kein Anwendungsname definiert.

47Z Es wurde KCOF=O angegeben, aber der 2. Parameterbereich fehlt oder ist ungültig.

48Z Ungültige Version der Datenstruktur.

55Z Der Eintrag in KCPI ist ungültig (beginnt nicht mit ">") oder die Vorgangs-Id wurde vom Auftraggeber-Vorgang bereits vergeben.

58Z Der Wert in KCOF ist ungültig.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z INIT-Aufruf fehlt oder im MSGTAC-Programm wurde die Ausprägung DM gegeben.

89Z Bei der Adressierung eines OSI TP-Auftragnehmers sind die nicht verwendeten Felder des KDCS-Parameterbereichs nicht mit binär null versorgt worden.

Eigenschaften des APRO-Aufrufs

- Ein Vorgang, der mit einer Partner-Anwendung über das OSI TP-Protokoll kommuniziert und dabei die Functional Unit "Commit" nutzt, kann nicht mit einer anderen Partner-Anwendung über LU6.1 kommunizieren.
- Wird in einem Vorgang, der an einer verteilten Transaktion beteiligt ist, das OSI TP-Protokoll verwendet, dann müssen alle Auftragnehmer dem gleichen lokalen ACCESS-POINT zugeordnet sein und dieser muss ggf. identisch zu dem ACCESS-POINT sein, der für die Kommunikation mit einem AG-Vorgang verwendet wird.
- Security-Typ P und S können Sie sowohl bei APRO DM als auch bei APRO AM wählen. Bei Dialog-Vorgängen wird der Auftraggeber beim Vorgangsstart unter der übergebenen Benutzerkennung angemeldet und bei Vorgangsende wieder abgemeldet. Bei Asynchron-Vorgängen ist die übergebene Benutzerkennung nur so lange aktiv, bis der Auftrag in die entsprechende Queue eingetragen ist.
  Bei Dialog-Vorgängen wird die Anmeldung des Auftraggebers unter der übergebenen Benutzerkennung abgelehnt, wenn unter dieser Benutzerkennung schon ein OSI TP Dialog-Vorgang läuft oder ein Benutzer über einen LTERM-Partner angemeldet ist und das mehrfache Anmelden eines Benutzers an die Anwendung verboten ist (SIGNON-Anweisung MULTI-SIGNON=NO) oder die Benutzerkennung mit RESTART=YES konfiguriert ist und nicht die Functional Unit "Commit" ausgewählt wurde.
  Um bei Asynchron-Vorgängen einen Auftrag in die entsprechende Queue einzuhängen, kann sich der Auftraggeber immer unter der übergebenen Benutzerkennung anmelden.

  Bei Auswahl von Security-Typ "S" übergibt openUTM die Benutzerkennung, unter der der lokale Vorgang läuft, an den Auftragnehmer-Vorgang. Als Datentyp wird T61 String angenommen.

  Wird Security-Typ "P" ausgewählt, müssen Benutzerkennung und Passwort angegeben werden. Zusätzlich muss angegeben werden, von welchem Datentyp Benutzerkennung und Passwort sind. Mögliche Datentypen sind Printable String, T61 String und Octet String. Die Wertebereiche dieser Datentypen sind im openUTM-Handbuch „Anwendungen generieren“ bei der KDCDEF-Anweisung LTAC beschrieben.

  Hinweis für BS2000-Systeme

  > Bei Angabe des Typs Octet String findet keine Code-Umsetzung statt. Der Typ Octet String ist bei Übertragung von Passwörtern notwendig, die als Hex-String (PASSWORD=X'aabbcc..') generiert wurden.
- Ein erfolgreicher APRO-DM-Aufruf bedeutet, dass eine logische Verbindung zur Partner-Anwendung besteht. Ein erfolgreicher APRO-Aufruf bedeutet aber noch nicht, dass ein Nachrichtenaustausch mit dem Auftragnehmer-Vorgang möglich ist. Eine Session bzw. Association wird erst am Ende desjenigen Teilprogrammlaufs belegt, in welchem die erste Nachricht an den Auftragnehmer-Vorgang ausgegeben wurde.
- Falls zum Zeitpunkt des APRO-Aufrufs noch keine Verbindung zur entfernten Anwendung besteht, dann leitet openUTM den Verbindungsaufbau ein.
- Gibt openUTM nach einem APRO DM-Aufruf einen KDCS-Returncode ! =000 zurück, so sollte anschließend keine Nachricht mit MPUT an den Auftragnehmer ausgegeben werden, da dann der Vorgang mit KCRCCC=74Z abgebrochen würde.
- Einem mit APRO DM adressierten Auftragnehmer-Vorgang muss in der gleichen Transaktion eine Nachricht gesendet werden, sonst bricht openUTM den Vorgang beim PEND mit KCRCCC=87Z ab.
- Einem mit APRO AM adressierten Asynchron-Vorgang muss vor dem nächsten PEND-Aufruf ein Asynchron-Auftrag zugeordnet werden, sonst bricht openUTM den Vorgang beim PEND mit 86Z ab.
- Ein RSET-Aufruf setzt die Adressierung eines Dialog-Vorgangs nur dann zurück, wenn der APRO DM im selben Verarbeitungsschritt erfolgte, d.h. es wurde noch keine Dialog-Nachricht an den Auftragnehmer-Vorgang (MPUT mit nachfolgendem PEND/PGWT).

- Existieren in einer Anwendung gleichzeitig mehrere Auftraggeber-Vorgänge, dann dürfen sie gleichnamige Vorgangs-Ids verwenden - auch zur Adressierung unterschiedlicher Auftragnehmer-Vorgänge.
- Soll ein Auftrags-Komplex an die ferne Anwendung gesendet werden, dann muss der APRO AM-Aufruf vor dem MCOM BC-Aufruf stehen und die Vorgangs-Id muss beim MCOM BC-Aufruf im Feld KCRN angegeben werden.
- Lebensdauer einer Vorgangs-Id

  Eine mit APRO DM erzeugte Vorgangs-Id ist ein sicherungsrelevantes Objekt, d.h., sie wird in die Transaktionssicherung einbezogen. Sie bleibt so lange erhalten, bis der Auftragnehmer-Vorgang beendet ist und wird erst am Ende der Transaktion freigegeben, in der der Auftraggeber-Vorgang die letzte Nachricht vom Auftragnehmer gelesen hat.

  Eine mit APRO AM erzeugte Vorgangs-Id wird freigegeben
  - nach erfolgreichem FPUT/DPUT NE-Aufruf,
  - beim nächsten RSET- oder PEND-Aufruf,
  - nach dem Returncode 40Z bei einem FPUT/DPUT-Aufruf, oder
  - bei Auftrags-Komplexen mit dieser Vorgangs-Id:
    Beim MCOM EC-Aufruf oder nach einem Returncode 40Z beim MCOM BC- oder bei einem DPUT-Aufruf.

  Wurde eine Vorgangs-Id freigegeben, dann kann sie für die nächste Auftraggeber/Auftragnehmer-Beziehung verwendet werden.

- Adressierung eines Master-LPAP

  Wenn mit dem APRO-Aufruf ein Master-LPAP adressiert wird, wird, falls es sich um den ersten APRO-Aufruf der aktuellen Transaktion für dieses Master-LPAP handelt, ein Slave-LPAP des LPAP-Bündel ausgewählt:
  - Bei einem APRO DM wird das erste Slave-LPAP ausgewählt, zu dem eine logische Verbindung aufgebaut ist. Ist zu keinem Slave-LPAP eine logische Verbindung aufgebaut, so ist der Rückgabewert 40Z/KD10.
  - Bei einem APRO AM wird das erste Slave-LPAP ausgewählt, zu dem eine logische Verbindung aufgebaut ist. Ist zu keinem Slave-LPAP eine logische Verbindung aufgebaut, so wird eines der Slave-LPAPs ausgewählt.

  Für jedes Slave-LPAP, das bei der Auswahl geprüft wird und zu dem keine Association oder Session aufgebaut ist, wird ein Association- oder Session-Aufbau angestoßen.

  Wenn ein Slave-LPAP ausgewählt wurde, wird bei jedem weiteren APRO-Aufruf in dieser Transaktion, der an das Master-LPAP gerichtet ist, dasselbe Slave-LPAP verwendet.

  Ein nachfolgender APRO DM kann den Returncode 40Z/KD10 liefern, wenn der erste APRO-Aufruf ein APRO AM war und zu keinem der Slave-LPAPs eine aufgebaute Association oder Session existierte, oder wenn die logische Verbindung zwischenzeitlich wieder abgebaut wurde.

> ¡ Detaillierte Informationen zur Verteilung von Nachrichten finden Sie auch im openUTM-Handbuch „Anwendungen generieren“.

7.4 CTRL Steuern eines OSI TP-Dialogs

Der Funktionsaufruf CTRL (control) findet Verwendung bei verteilter Verarbeitung über das OSI TP-Protokoll. Er dient dazu, einen Dialog mit einem OSI TP-Partner explizit zu steuern. Die Aufrufe CTRL PR und CTRL PE dürfen nur an Auftragnehmer-Vorgänge gerichtet werden, für die die Functional Unit "Commit" ausgewählt wurde.

Es gibt mehrere Ausprägungen des Aufrufs CTRL:

- CTRL PR (Prepare to Commit)
  Mit CTRL PR wird der Auftragnehmer-Vorgang aufgefordert, das Transaktionsende einzuleiten. Wenn der lokale Vorgang zusätzlich mit einem MPUT-Aufruf Daten an den Partner sendet, kann er im CTRL-Aufruf steuern, ob der ferne Vorgang noch Daten senden darf.
- CTRL PE (Prepare End Dialogue)
  Mit CTRL PE wird der Auftragnehmer-Vorgang aufgefordert, das Dialogende einzuleiten. Wenn der lokale Vorgang zusätzlich mit einem MPUT-Aufruf Daten an den Partner sendet, kann er im CTRL-Aufruf steuern, ob der ferne Vorgang noch Daten senden darf.
- CTRL AB (Abort Dialogue)
  Mit CTRL AB wird für den Dialog mit einem Auftragnehmer-Vorgang eine abnormale Beendigung eingeleitet. Ein MPUT-Aufruf an diesen Auftragnehmer wird nicht gesendet sondern gelöscht. Ist für den Dialog die Commit Funktionalität ausgewählt, dann sorgt openUTM dafür, dass die verteilte Transaktion auf den letzten Sicherungspunkt zurückgesetzt wird; erst dann wird der Dialog beendet. Für jeden abnormal zu beendenden Dialog ist ein eigener CTRL Aufruf erforderlich.

Versorgen des 1. Parameters

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dazu notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOM	KCLA	KCLM	KCRN	KCMF/kcfn	KCNORPLY
Prepare to Commit	"PR"	0	0	Vorgangs-Id	Leerzeichen	"Y"/0
Prepare End Dialogue	"PE"	0	0	Vorgangs-Id	Leerzeichen	"Y"/0
Abort Dialogue	"AB"	0	0	Vorgangs-Id	Leerzeichen	0

Versorgen des 2. Parameters

Hier müssen Sie die Adresse des Nachrichtenbereichs angeben. Der Nachrichtenbereich wird in dieser Version von openUTM nicht verwendet, er ist für spätere Erweiterungen vorgesehen.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"CTRL"
KCOM	"PR"/"PE"/"AB"
KCLA	0
KCLM	0
KCRN	Vorgangs-Id
KCMF/kcfn	Leerzeichen
KCNORPLY	"Y"/0

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_CTRLPR/KDCS_CTRLPE/KDCS_CTRLAB	(kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich sind für den CTRL-Aufruf folgende Eintragungen notwendig:

KCOP

In das Feld KCOP muss der Operationscode CTRL eingetragen werden.

KCOM

Das Feld KCOM muss eine der folgenden Operationsmodifikationen enthalten:

- **PR** (PRepare to commit)
  Mit dieser Ausprägung wird ein Auftragnehmer-Vorgang aufgefordert, das Transaktionsende einzuleiten. Außerdem gilt:
  - Wenn der lokale Vorgang zusätzlich mit einem MPUT-Aufruf Daten an den Auftragnehmer-Vorgang sendet, kann er mit dem Feld KCNORPLY steuern, ob der Auftragnehmer-Vorgang in der laufenden Transaktion noch Daten senden darf. Der lokale Vorgang wird nach Ende des Verarbeitungsschritts erst dann fortgesetzt, wenn der ferne Vorgang das Transaktionsende eingeleitet hat.
  - Wenn der lokale Vorgang keine Daten an den Auftragnehmer-Vorgang sendet, dann darf der Auftragnehmer-Vorgang in dieser Transaktion ebenfalls keine Daten mehr senden. Der lokale Vorgang wartet dann nach Ende des
    Verarbeitungsschritts nicht darauf, dass der Auftragnehmer-Vorgang das Transaktionsende einleitet.

- **PE** (Prepare End dialogue)
  Mit dieser Ausprägung wird ein AN-Vorgang aufgefordert, das Dialogende einzuleiten. Außerdem gilt:
  - Wenn der lokale Vorgang zusätzlich mit einem MPUT-Aufruf Daten an den Auftragnehmer-Vorgang sendet, kann er mit dem Feld KCNORPLY steuern, ob der Auftragnehmer-Vorgang im laufenden Dialog noch Daten senden darf. Der lokale Vorgang wird nach Ende des Verarbeitungsschritts erst dann fortgesetzt, wenn der ferne Vorgang das Dialogende eingeleitet hat.
  - Wenn der lokale Vorgang keine Daten an den Auftragnehmer-Vorgang sendet, dann darf der Auftragnehmer-Vorgang in diesem Dialog ebenfalls keine Daten mehr senden. Der lokale Vorgang wartet dann nach Ende des
    Verarbeitungsschritts nicht darauf, dass der Auftragnehmer-Vorgang das Dialogende einleitet.
- **AB** (ABort dialogue)
  Mit dieser Ausprägung wird für einen Dialog mit einem AN-Vorgang eine abnormale Beendigung eingeleitet. Ist für den Dialog die Commit Funktionalität ausgewählt, dann erzwingt openUTM, dass die verteilte Transaktion auf den letzten
  Sicherungspunkt durch einen geeigneten PEND-Aufruf zurückgesetzt wird; erst dann wird der Dialog beendet. Für jeden abnormal zu beendenden Dialog ist ein eigener CTRL Aufruf erforderlich. Eine mit MPUT ausgegebene Nachricht für den Partner wird beim CTRL AB gelöscht.

KCLA

Das Feld KCLA muss mit null versorgt werden.

KCLM

Das Feld KCLM muss mit null versorgt werden.

KCRN

Im Feld KCRN ist die Vorgangs-Id (VGID) des Partner-Vorgangs anzugeben, auf den sich der CTRL Aufruf bezieht.

KCMF/kcfn

Im Feld KCMF/kcfn sind Leerzeichen anzugeben.

KCNORPLY

Im Feld KCNORPLY kann bei KCOM=PR/PE der Wert Y eingetragen werden. Mit diesem Wert wird dem Auftragnehmer-Vorgang angezeigt, dass er in dieser Transaktion bzw. diesem Dialog keine Daten mehr senden darf, auch wenn der lokale Vorgang noch Daten mit MPUT an den Auftragnehmer-Vorgang sendet.

Alle nicht verwendeten Felder müssen mit binär null versorgt werden.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   die Adresse des KDCS-Parameterbereichs.

2. Parameter

   die Adresse des Nachrichtenbereichs. Diese Adresse muss bei allen CTRL-Aufrufen angegeben werden, obwohl CTRL-Aufrufe derzeit den Nachrichtenbereich nicht verwenden.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM siehe openUTM-Handbuch „Meldungen, Test und Diagnose“.

KDCS-Returncodes im Feld KCRCCC beim CTRL-Aufruf

Im Programm sind auswertbar:

000	Die Funktion wurde ordnungsgemäß durchgeführt.
40Z	Die Anwendung wurde ohne verteilte Verarbeitung generiert.
41Z	Der Aufruf CTRL ist an dieser Stelle unzulässig. Mögliche Gründe: • Der Aufruf wurde für einen Asynchron-Vorgang gegeben; d.h. die angegebene Vorgangs-Id wurde mit einem APRO AM-Aufruf definiert. • Der Aufruf ist an einen Partner gerichtet, an den zuvor ein MPUT HM abgesetzt wurde. • Ein CTRL PE oder PR ist an einen Partner gerichtet, für den die Commit FU nicht ausgewählt wurde. • Ein CTRL PE oder PR ist an einen Partner gerichtet, an den noch kein MPUT nach dem APRO gegeben wurde.
42Z	Die Funktionsausprägung in KCOM ist ungültig.
43Z	Der in KCLA oder KCLM angegebene Wert ist ungültig.
44Z	Die in KCRN angegebene Vorgangs-Id ist ungültig oder es wurde keine Vorgangs-Id angegeben.
45Z	Das Feld KCMF/kcfn wurde nicht mit Leerzeichen versorgt.
49Z	Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

54Z Das Feld KCNORPLY hat bei CTRL PR oder PE einen Wert ungleich Y oder binär null.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z In dem Teilprogrammlauf wurde noch kein INIT-Aufruf gegeben.

77Z Ungültige Bereichsadresse.

Eigenschaften des CTRL-Aufrufs

- Zum Ende eines Verarbeitungsschritts, in dem ein CTRL PR oder PE Aufruf und ein MPUT Aufruf an ein und denselben Partner gegeben wurde, darf kein Transaktionsende angefordert werden.
- Der Aufruf CTRL PR / PE / AB darf nur an solche AN-Vorgänge gerichtet werden, mit denen ein verteilter Dialog geführt wird, d.h. die mit einem APRO DM adressiert wurden.
- Die Aufrufe CTRL PR und CTLR PE dürfen nur an solche AN-Vorgänge gerichtet werden, für welche die Commit Functional Unit ausgewählt wurde und für die nach dem APRO bereits ein MPUT gegeben wurde.
- In einem Verarbeitungsschritt bzw. Teilprogrammlauf dürfen CTRL-Aufrufe für mehrere Partner gegeben werden.
- Nach einem CTRL AB Aufruf für einen Dialog mit einem AN-Vorgang, bei dem die Commit Funktionalität ausgewählt ist, sind nur die PEND-Aufrufe mit den Operationsmodifikationen RS, FR und ER erlaubt.

7.5 DADM Administration von Message Queues

Der Aufruf DADM (delayed free message administration) bietet folgende Funktionen zur Administration von Message Queues:

- Übersichtsinformationen über Nachrichten einer Message Queue in den Nachrichtenbereich lesen (Benutzerkennung, Auftrags-Identifikation (Auftrags-Id), Zeitpunkt der Erzeugung, Startzeit und vorhandene Quittungsaufträge, ursprüngliches Ziel ...)
- Benutzerinformationen, die mit DPUT NI/QI/+I/-I erzeugt wurden, in den Nachrichtenbereich lesen
- die Bearbeitungsreihenfolge von Nachrichten einer Message Queue ändern
- einzelne Nachrichten der Message Queue oder auch alle Nachrichten der Message Queue löschen
- einzelne oder alle Nachrichten der Dead Letter Queue wieder ihrem jeweiligen ursprünglichen Ziel oder einem neuen Ziel zuordnen

Unix-, Linux- und Windows-Systeme
In einer UTM-Cluster-Anwendung können Sie nur Message Queues der lokalen Knoten-Anwendung administrieren.

Im Folgenden wird das Format des DADM-Aufrufs ausführlich dargestellt. Weiterführende Informationen zur Administration von Message Queues finden Sie im openUTM-Handbuch „Anwendungen administrieren“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich							
	KCOP	KCOM	KCLA	KCRN	KCLT	KCQTYP	KCMOD	Zeitpunkt [1]
Übersichtsinformation lesen	"DADM"	"RQ"	54	Auftrags-Id /Leerzeichen	LTERM/(OSI-) LPAP/TAC	binär null	binär null	binär null
					Queue	"Q"/"U"		
Benutzerinformation lesen	"DADM"	"UI"	Länge	Auftrags-Id	binär null	binär null	binär null	Zeitpunkt (absolut)
Reihenfolge ändern	"DADM"	"CS"	0	Auftrags-Id	binär null	binär null	binär null	Zeitpunkt (absolut)
Löschen einer einzelnen Nachricht einer Queue	"DADM"	"DL"	0	Auftrags-Id	LTERM/(OSI-) LPAP/TAC	binär null	"C"/"N"	Zeitpunkt (absolut)
					Queue	"Q"/"U"		
Löschen aller Nachrichten einer Queue	"DADM"	"DA"	0	Leerzeichen	LTERM/(OSI-) LPAP/TAC	binär null	binär null	binär null
					Queue	"Q"/"U"		
Einzelne Nachricht verschieben	"DADM"	"MV"	0	Auftrags-Id	TAC/(OSI-)LPAP /Leerzeichen	binär null	binär null	Zeitpunkt (absolut)
Alle Nachrichten verschieben	"DADM"	"MA"	0	Leerzeichen	TAC/(OSI-)LPAP /Leerzeichen	binär null	binär null	binär null

[1] Der Zeitpunkt wird in den Feldern KCTAG/kcday, KCSTD/kchour und KCMIN angegeben.

Alle nicht verwendeten Felder des KDCS-Parameterbereichs müssen mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, in den openUTM die Nachricht lesen soll. Für die Strukturierung des Nachrichtenbereichs beim Aufruf DADM RQ steht eine Sprach-spezifische Datenstruktur zur Verfügung, für COBOL das COPY-Element KCDADC und für C/C++ die Include-Datei *kcdad.h*.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"DADM"
KCOM	"RQ"/"UI"/"CS"/"DL"/"DA"/"MV"/"MA"
KCLA	Länge in Byte/0
KCRN	Auftrags-Id/Leerzeichen
KCLT	LTERM-Name/(OSI-)LPAP-Name/TAC/Queue/binär null/Leerzeichen
KCQTYP	Typ des Ziels:"Q"/"U"/binär null
KCMOD	"C"/"N"/binär null
KCTAG/...	Zeitangabe (absolut)
• KCTAG/kcday	• Tag (absolut)/binär null
• KCSTD/kchour	• Stunde (absolut)/binär null
• KCMIN	• Minute (absolut)/binär null
• KCSEK/kcsec	• Sekunde (absolut)/binär null

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_DADMRQ	(nb,kcla,kcrn,kclt)
KDCS_QADMRQ	(nb,kcla,kcrn,kclt,kcqtyp)
KDCS_DADMUI	(nb,kcla,kcrn,kcday,kchour,kcmin,kcsec)
KDCS_DADMCS	(nb,kcrn,kcday,kchour,kcmin,kcsec)
KDCS_DADMDL	(nb,kcrn,kclt,kcmod,kcday,kchour,kcmin,kcsec)
KDCS_QADMDL	(nb,kcrn,kclt,kcmod,kcday,kchour,kcmin,kcsec,kcqtyp)
KDCS_DADMDA	(nb,kclt)
KDCS_QADMDA	(nb,kclt,kcqtyp)
KDCS_DADMMV	(nb,kcrn,kclt,kcday,kchour,kcmin,kcsec,kcqtyp)
KDCS_DADMMA	(nb,kclt,kcqtyp)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	**Inhalt**
KCRLM	tatsächliche Länge
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Auftrags-Id/Leerzeichen

In den KDCS-Parameterbereich tragen Sie für den DADM-Aufruf Folgendes ein:

KCOP

Im Feld KCOP geben Sie den Operationscode DADM an.

KCOM

Im Feld KCOM wählen Sie die Operationsmodifikation:

- RQ zum Lesen der Übersichtsinformationen zu einer Message Queue
- UI zum Lesen einer durch DPUT NI/QI erzeugten Benutzerinformation eines Hauptauftrags
- CS um eine bestimmte Nachricht vorzuziehen

- DL zum Löschen einer einzelnen Nachricht
- DA zum Löschen aller Nachrichten in der Message Queue
- MV zum Verschieben einer einzelnen Nachricht der Dead Letter Queue in die ursprüngliche Message Queue oder an einen beliebigen Asynchron-TAC, eine beliebige TAC-Queue, einen LPAP-Partner oder einen OSI-LPAP-Partner.
- MA zum Verschieben aller Nachrichten der Dead Letter Queue. Dabei gibt es zwei Möglichkeiten:
  - Verschieben in die jeweiligen ursprünglichen Message Queues
  - Verschieben aller Nachrichten mit ursprünglichem Ziel TAC, LPAP oder OSI-LPAP an ein beliebiges anderes Ziel vom gleichen Typ (Asynchron-TAC/TAC-Queue, LPAP-Partner oder OSI-LPAP-Partner). Nachrichten, deren ursprüngliches Ziel nicht vom Typ des neuen Ziels ist, verbleiben in der Dead Letter Queue.
    Um alle Nachrichten zu verschieben, sind also bis zu drei Aufrufe mit neuen Zielen vom Typ TAC, LPAP und OSI-LPAP erforderlich.

KCLA

Im Feld KCLA geben Sie die Länge an, in der die Daten in den Nachrichtenbereich übertragen werden sollen. Bei KCOM = RQ wird 54, bei KCOM = CS/DL/DA/MV/MA wird 0 eingetragen.

KCRN

Im Feld KCRN identifizieren Sie die Nachricht der Queue, die administriert werden soll. Folgende Angaben sind notwendig:

- Leerzeichenbei KCOM = DA/MA oder wenn sich der Aufruf bei KCOM = RQ auf die erste Nachricht in der Queue bezieht.
- die Auftrags-Idbei KCOM = UI/CS/DL/MV oder wenn sich der Aufruf bei KCOM = RQ auf nachfolgende Nachrichten in der Queue bezieht. Die Auftrags-Id wird immer beim vorigen DADM RQ im Feld KCRMF /kcrfn zurückgegeben.

KCLT

Im Feld KCLT identifizieren Sie die Queue, d.h.

- Bei KCOM = RQ/DL/DA geben Sie an:
  - den LTERM-Namen, wenn die Nachricht an einen LTERM-Partner gerichtet war,
  - den (OSI-)LPAP-Namen, wenn die Nachricht an einen (OSI-)LPAP-Partner gerichtet war,
  - den TAC, wenn die Nachricht an ein Asynchron-Programm gerichtet war,
  - den Namen der Queue, wenn Sie Nachrichten einer USER-Queue, einer TAC-Queue oder einer Temporären Queue administrieren wollen.

- Bei KCOM = UI/CS geben Sie binär null an.
- Bei KCOM = MV/MA geben Sie an:
  - den TAC, wenn die einzelne bzw. alle Nachrichten mit ursprünglichem Ziel TAC oder TAC-Queue an ein Asynchron-Programm gerichtet werden sollen,
  - den Namen einer TAC-Queue, wenn die einzelne bzw. alle Nachrichten mit ursprünglichem Ziel TAC oder TAC-Queue an eine Service-gesteuerte Queue gerichtet werden sollen,
  - den Namen eines LPAP-Partners (aber kein MASTER-LU61-LPAP), wenn die einzelne bzw. alle Nachrichten mit ursprünglichem Ziel LPAP an einen LPAP-Partner gerichtet werden sollen,
  - den Namen eines OSI-LPAP-Partners (aber kein MASTER-OSI-LPAP), wenn die einzelne bzw. alle Nachrichten mit ursprünglichem Ziel OSI-LPAP an einen OSI-LPAP-Partner gerichtet werden sollen,
  - Leerzeichen, wenn die einzelne bzw. alle Nachrichten wieder ihrem jeweiligen ursprünglichen Ziel zugeordnet werden sollen.

> **i** Alle Nachrichten, deren ursprüngliches Ziel nicht zum neuen Ziel passt, verbleiben in der Dead-Letter-Queue.

KCQTYP

Im Feld KCQTYP geben Sie den Typ der Queue an:

- bei KCOM = RQ/DL/DA geben Sie an:
  - binär null, wenn die Queue zu einem LTERM, einem (OSI-)LPAP oder einem TAC gehört oder wenn es eine TAC-Queue ist,
  - Q bei einer mit QCRE erzeugten Temporären Queue,
  - U bei einer USER-Queue.
- bei KCOM = UI/CS/MV/MA geben Sie binär null an.

KCMOD

Im Feld KCMOD geben Sie beim Löschen eines Auftrags-Komplexes an, ob openUTM den negativen Quittungsauftrag aktivieren soll. Folgende Werte sind möglich:

- binär null bei KCOM = RQ/UI/CS/DA/MV/MA,
- C zum Löschen des kompletten Auftrags bei KCOM = DL; bei Auftrags-Komplexen werden alle Quittungsaufträge ebenfalls gelöscht,
- N zum Aktivieren des negativen Quittungsauftrags bei KCOM = DL; die Nachricht selbst wird gelöscht

KCTAG/...

- Bei KCOM = UI/CS/DL/MV geben Sie in diesen Feldern den Zeitpunkt an, an dem die Nachricht erzeugt wurde, und zwar im Feld **KCTAG/kcday** den Tag im Jahr (Industrietag, Wert von 001 bis 366), in **KCSTD /kchour** die Stunde, in **KCMIN** die Minute und in **KCSEK/ kcsec** die Sekunde. Dieser Zeitpunkt kann vorher mit DADM RQ ermittelt werden.
- Bei KCOM = RQ/DA/MA wird binär null eingetragen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht lesen soll. Diese Adresse müssen Sie auch angeben, wenn Sie in KCLA 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

bei KCOM = RQ/UI im angegebenen Nachrichtenbereich die Nachricht in der tatsächlichen, höchstens aber in der angeforderten Länge.

KCRLM

im Feld KCRLM die tatsächliche Länge der Nachricht, ggf. in Abweichung von der angeforderten Länge in KCLA des Parameterbereichs.

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

im Feld KCRMF/kcrfn bei KCOM = RQ die Auftrags-Id der nächsten Nachricht in der Queue (siehe KCLT) oder Leerzeichen bei der letzten Nachricht der Queue.

KDCS-Returncodes im Feld KCRCCC beim DADM-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde ausgeführt (bei KCOM = RQ/UI) bzw. der Administrationsauftrag wurde angenommen (bei KCOM = CS/DL/DA/(MV/MA).
Über die tatsächliche Ausführung wird erst beim Transaktionsende entschieden (siehe „Eigenschaften des DADM-Aufrufs“).

01Z Längenkonflikt: KCLA < KCRLM, die Nachricht wurde abgeschnitten.

07Z Es konnten nicht alle Nachrichten der Dead Letter Queue verschoben werden, weil der taskspezifische Pufferbereich für die Wiederanlauf-Information zu klein generiert wurde, siehe openUTM-Handbuch „Anwendungen generieren“ "Wiederanlaufbereich".
Maßnahme: Generierung mit KDCDEF, Pufferbereich mit MAX RECBUF=(...,length) größer definieren oder den DADM-Aufruf wiederholen.

40Z Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, ursprüngliches Ziel nicht mehr existent, keine Administrationsberechtigung, Blockade durch anderen Vorgang), siehe KCRCDC.

42Z Der Eintrag in KCOM ist ungültig.

43Z Die Längenangabe in KCLA ist negativ bzw. ungültig.

44Z Die in KCRN angegebene Auftrags-Id ist ungültig.

46Z Die Angabe in KCLT ist ungültig. Mögliche Ursachen:

- der angegebene LTERM- oder (OSI-)LPAP-Name ist nicht vorhanden
- der TAC ist ungültig oder gesperrt
- der TAC ist kein Vorgangs-TAC oder ein Dialog-TAC
- es gibt keine USER-Queue oder Temporäre Queue mit dem angegebenen Namen oder der in KCTYP angegebene Typ passt nicht zu dem Queue-Namen
- bei KCOM=MV/MA: Es wurde ein LTERM-Name, Master-LU61-LPAP-Name oder MASTER-OSI-LPAP-Name angegeben oder der angegebene TAC wurde gelöscht oder es wurde KDCMSGTC oder KDCDLETQ angegeben.
- bei KCOM=MV: Es wurden Leerzeichen angegeben, aber das ursprüngliche Ziel der Nachricht wurde gelöscht.
- bei KCOM=MV: Das ursprüngliche Ziel und das neue Ziel der Nachricht passen vom Typ her nicht zusammen, d.h.
  - ursprünglich TAC oder TAC-Queue, neu LPAP oder OSI-LPAP
  - ursprünglich LPAP, neu TAC oder TAC-Queue oder OSI-LPAP
  - ursprünglich OSI-LPAP, neu TAC oder TAC-Queue oder LPAP

47Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

56Z Der Wert in KCMOD oder die Zeitangabe in KCTAG/kcday, KCSTD/kchour, KCMIN oder KCSEK/kcsec ist ungültig.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Teilprogramm-Lauf wurde noch kein INIT-Aufruf gegeben.

Rückgaben im Nachrichtenbereich bei DADM RQ

Für die Rückgabeninformation bei Aufruf DADM RQ steht eine Datenstruktur zur Verfügung, für COBOL das COPY-Element KCDADC und für C/C++ die Include-Datei *kcdad.h*. Diese Datenstruktur können Sie zur Definition des Nachrichtenbereichs verwenden. Sie hat folgenden Aufbau:

Byte	Feldname COBOL/C/C++	Bedeutung	
1 - 8	KCDAGUS	UTM-Benutzerkennung des Auftraggebers	
9 - 16	KCDADPID	Auftrags-Id (von openUTM vergeben)	
17 - 25	KCDAGTIM[1]	Zeitpunkt des FPUT-/DPUT-Aufrufs in der Form *dddhhmmss*:	
17 - 19	KCDAGDOY	*ddd*	Tag im Jahr (Wertebereich 001 - 366)
20 - 21	KCDAGHR	*hh*	Stunde (Wertebereich 00 - 23)
22 - 23	KCDAGMIN	*mm*	Minute (Wertebereich 00 - 59)
24 - 25	KCDAGSEC	*ss*	Sekunde (Wertebereich 00 - 59)
26 - 34	KCDASTIM[1]	bei zeitverzögertem Auftrag die gewünschte Startzeit in der Form *dddhhmmss*:	
26 - 28	KCDASDOY	*ddd*	Tag im Jahr (Wertebereich 001 - 366)
29 - 30	KCDASHR	*hh*	Stunde (Wertebereich 00 - 23)
31 - 32	KCDASMIN	*mm*	Minute (Wertebereich 00 - 59)
33 - 34	KCDASSEC	*ss*	Sekunde (Wertebereich 00 - 59)
		Bei einem Auftrag ohne Zeitverzögerung werden Leerzeichen zurückgegeben.	
35	KCDAPMSG	Y	Positiver Quittungsauftrag vorhanden
		N	Kein positiver Quittungsauftrag vorhanden
36	KCDANMSG	Y	Negativer Quittungsauftrag vorhanden
		N	Kein negativer Quittungsauftrag vorhanden
37 - 44	KCDADEST	Ziel der Nachricht bzw. ursprüngliches Ziel bei Dead Letter Queue	

Byte	Feldname COBOL/C/C++	Bedeutung	
45	KCDATYPE	Typ des Ziels der Nachricht bzw. Typ des ursprünglichen Ziels bei Dead Letter Queue:	
		Q	für temporäre Queue
		U	für USER-Queue
		T	für TAC-Queue
		A	für Asynchron-TAC
		L	für LTERM
		P	für LPAP
		O	für OSI-LPAP
46 - 53	KCDAFCTM	Erzeugungs- oder Umwandlungszeitpunkt von DPUT in FPUT der Nachricht. Bei Service-gesteuerten Queues können die mit DADM RQ angezeigten Nachrichten den mit DGET BF gelesenen Nachrichten eindeutig zugeordnet werden. KCDADPID bzw. KCDAFCTM müssen mit den Rückgabewerten KCRDPID bzw. KCRGTM des entsprechenden DGET BF Aufrufs übereinstimmen.	
54	KCDAGUST	Typ des Auftraggebers:	
		U	für USER
		S	für LU6.1-Session
		O	für OSI-Association

[1] Für C/C++ sind die zusammenfassenden Felder KCDAGTIM und KCDASTIM nicht definiert, wohl aber die spezifischen Felder für Tag/Stunde/Minute/Sekunde.

Eigenschaften des DADM-Aufrufs

- Ermitteln der Auftrags-Id

  Die Auftrags-Id wird von openUTM intern vergeben. Sie kann im Teilprogramm wie folgt mit DADM RQ-Aufrufen ermittelt werden: Beim ersten DADM RQ-Aufruf (mit KCRN = Leerzeichen) informiert openUTM über die erste Nachricht in der Queue. openUTM schreibt dabei u.a. die Auftrags-Id in den Nachrichtenbereich. Stehen für die Queue weitere Nachrichten in der Message Queue, dann gibt openUTM in KCRMF/kcrfn die Auftrags-Id der jeweils nächsten Nachricht zurück. Bei der letzten Nachricht in der Queue trägt openUTM im Feld KCRMF/kcrfn Leerzeichen ein.
- DADM CS/DL/DA/MV/MA-Aufrufe (Reihenfolge ändern, löschen oder verschieben) unterliegen der Transaktionssicherung und werden deshalb erst bei Transaktionsende ausgeführt. Daher garantiert KCRCCC = 000 nach einem solchen Aufruf noch nicht, dass der Aufruf erfolgreich ausgeführt werden kann, denn in der Zwischenzeit könnte die Nachricht in der Queue durch einen anderen DADM-Aufruf gelöscht worden sein. Ob ein DADM CS/DL/DA/MV/MA-Aufruf erfolgreich war, kann man in einer nachfolgenden Transaktion durch einen DADM RQ-Aufruf erfahren.

- Durch den Aufruf DADM CS wird die entsprechende Nachricht an die erste Stelle der zugehörigen Warteschlage vorgezogen. Für zeitgesteuerte Nachrichten, deren Startzeitpunkt noch nicht erreicht ist, wird der Aufruf mit 40Z abgewiesen.
- Nach einem Löschauftrag mit DADM DL/DA darf innerhalb derselben Transaktion kein weiterer Löschauftrag und auch kein DADM CS-Aufruf gegeben werden; openUTM lehnt dies mit 40Z ab.
- Ein Teilprogramm, das einen DADM-Aufruf absetzt, muss unter einer administrationsberechtigten Benutzerkennung ablaufen, sonst quittiert openUTM den DADM-Aufruf mit 40Z. Allerdings gibt es folgende Ausnahme:
  Wenn ein Vorgang von einem Druckersteuer-LTERM aus gestartet wurde und wenn nur solche Aufträge administriert werden, die an Drucker dieses Druckersteuer-LTERMs gerichtet sind, dann benötigen weder das Teilprogramm noch der Benutzer die Administrationsberechtigung.
- Die Dead Letter Queue besteht aus Nachrichten, die nicht verarbeitet werden konnten. Um diese Nachrichten evtl. nach einer Fehlerbehebung noch verarbeiten zu können, müssen sie entweder ihrem ursprünglichen Ziel oder einem neuen Ziel vom gleichen Typ zugeordnet werden.
  Mit DADM MV kann eine einzelne Nachricht, mit DADM MA können alle Nachrichten der Dead Letter Queue verschoben werden. Dabei gibt es zwei Möglichkeiten:
  - Verschieben in die jeweilige(n) ursprüngliche(n) Message Queue(s).
  - Verschieben der Nachricht bzw. aller Nachrichten mit ursprünglichem Ziel TAC (Asynchron-TAC oder TAC-Queue), LPAP oder OSI-LPAP an ein beliebiges anderes Ziel vom gleichen Typ.

  Dabei wird ein evtl. definiertes QLEV und der STATUS der Empfänger-Queue ignoriert.

  Bei DADM MA verbleiben Nachrichten in folgenden Fällen in der Dead Letter Queue, ohne dass dies über einen Returncode angezeigt wird:
  - KCLT = Leerzeichen: Alle Nachrichten, deren ursprüngliches Ziel nicht mehr existiert.
  - KCLT != Leerzeichen: Alle Nachrichten, deren ursprüngliches Ziel nicht zum neuen Ziel passt. Nachrichten, deren ursprüngliches Ziel nicht mehr existiert, passen dabei nur zum neuen Ziel Asynchron-TAC oder TAC-Queue.

7.6 DGET Lesen einer Nachricht aus einer Service-gesteuerten Queue

Mit dem Aufruf DGET (data GET) wird eine Nachricht oder Teilnachricht aus einer Service-gesteuerten Queue in den Nachrichtenbereich eingelesen.
Service-gesteuerte Queues sind TAC-Queues, USER-Queues und Temporäre Queues.

DGET bietet mehrere Varianten, um Nachrichten einer Queue zu lesen:

- Sequenzielles Verarbeiten (DGET FT/NT)
- Browsen (DGET BF/BN)
- Gezieltes Verarbeiten (DGET PF/PN)

Beim sequenziellen oder gezielten Verarbeiten wird die Nachricht gelesen und anschließend aus der Queue gelöscht, beim Browsen bleibt sie in der Queue. Für die Dead Letter Queue ist nur Browsen (Lesen ohne Löschen) erlaubt.

Mit jeder Variante können auch mehrere Teilnachrichten gelesen werden. Der erste Teil wird mit DGET FT/BF/PF (First) gelesen. Die weiteren Teile werden innerhalb desselben Teilprogrammlaufs mit der Next-Variante DGET NT /BN/PN gelesen, ohne zwischenzeitlichen PGWT-Aufruf.

Es können so viel Nachrichtenteile gelesen werden, wie Nachrichtenteile mit DPUT QT geschickt wurden. Jede mit DPUT QT gesendete Teilnachricht muss mit einem eigenen DGET gelesen werden. Nicht gelesene Teilnachrichten gehen verloren, wenn

- mit DGET FT eine neue Nachricht gelesen wird,
- PGWT aufgerufen wird,
- der Teilprogrammlauf beendet wird.

Nach dem Rücksetzen der Transaktion werden verarbeitete Nachrichten wieder in die Queue gestellt (Redelivery) und können damit erneut mit DGET gelesen und verarbeitet werden. Die maximale Anzahl der Zustellungen lässt sich per Generierung einstellen (MAX REDELIVERY=). Ist diese Grenze erreicht, wird die verarbeitete Nachricht abhängig vom Parameter DEAD-LETTER-Q der TAC-Anweisung der Generierung zum Transaktionsende entweder gelöscht oder in die Dead Letter Queue gesichert, sofern (bei Message-Komplexen) kein negativer Quittungsauftrag definiert wurde.
Nachrichten aus USER- oder temporären Queues können nicht in die Dead Letter Queue gesichert werden. Sie gehen also nach maximaler Anzahl der Zustellungen verloren.

Im Folgenden wird das Format des DGET-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Nachrichten-Queues" finden Sie im Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgenden Tabellen zeigen die verschiedenen Möglichkeiten und die notwendigen Angaben im KDCS-Parameterbereich.

Sequenzielles Verarbeiten									
Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**								
	KCOP	**KCOM**	**KCLA**	**KCMF/ kcfn**	**KCRN**	**KCQTYP**	**KCWTIME**	**KCQRC**	**KCDPID**
Neue Nachricht/ erste Teilnachricht der ersten Nachricht verarbeiten	"DGET"	"FT"	Länge	Leerzeichen	Queue-Name	Typ der Queue	Sekunden Wartezeit	0	binär null
Nächste Teilnachricht verarbeiten	"DGET"	"NT"	Länge	Leerzeichen	Queue-Name	Typ der Queue	0	0	binär null

Browsen									
Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**								
	KCOP	**KCOM**	**KCLA**	**KCGTM**	**KCRN**	**KCQTYP**	**KCWTIME**	**KCQRC**	**KCDPID**
Erste Teilnachricht der ersten bzw. nächsten Nachricht lesen	"DGET"	"BF"	Länge	Leerzeichen /Erzeugungszeit*	Queue-Name	Typ der Queue	Sekunden Wartezeit	-1 /Redelivery-Zähler	Leerzeichen /DPUT-ID
Nächste Teilnachricht lesen	"DGET"	"BN"	Länge	Erzeugungszeit*	Queue-Name	Typ der Queue	0	0	DPUT-ID

Gezieltes Verarbeiten									
Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**								
	KCOP	**KCOM**	**KCLA**	**KCGTM**	**KCRN**	**KCQTYP**	**KCWTIME**	**KCQRC**	**KCDPID**
Erste Teilnachricht einer bestimmten Nachricht verarbeit.	"DGET"	"PF"	Länge	Erzeugungszeit*	Queue-Name	Typ der Queue	0	0	DPUT-ID
Nächste Teilnachricht verarbeiten	"DGET"	"PN"	Länge	Erzeugungszeit*	Queue-Name	Typ der Queue	0	0	DPUT-ID

* Wert aus dem KB-Rückgabefeld KCRGTM des vorhergehenden DGET BF

Versorgen des 2. Parameters

Hier müssen Sie die Adresse des Nachrichtenbereichs bereitstellen, in den openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"DGET"
KCOM	"FT"/"NT"/"BF"/"BN"/"PF"/"PN"
KCLA	Länge in Byte
KCMF/kcfn KCGTM	Leerzeichen Leerzeichen/Erzeugungszeit
KCRN	Name der Queue
KCQTYP	"T"/"U"/"Q"
KCWTIME	Wartezeit in Sekunden/0
KCQRC	0/-1/Redelivery-Zähler
KCDPID	Binär null / Leerzeichen / DPUT-ID

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_DGETFT/KDCS_DGETNT	(nb,kcla,kcfn,kcrn,kcqtyp,kcwtime,kcfkt1, kcfkt2)
KDCS_DGETBF/KDCS_DGETBN/KDCS_DGETPF /KDCS_DGETPN	(nb,kcla,kcrn,kcqtyp,kcwtime,kcqrc,kcgtm, kcdpid)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Länge
KCRMF/kcrfn (nur DGET FT/NT)	bei OSI TP-Partner: Name der abstrakten Syntax
KCRWVG (nur DGET FT)	Anzahl der Vorgänge, die schon warten
KCRUS (nur DGET FT)	UTM-Benutzerkennung des Nachrichten-Erzeugers
KCRQRC (nur DGET BF)	Queue-spezifischer Redelivery-Zähler
KCRGTM (nur DGET BF)	Erzeugungszeit der gelesenen Nachricht
KCRDPID (nur DGET BF)	DPUT-ID der gelesenen Nachricht
KCRRC (nur DGET FT/BF/BN/PF)	Redelivery-Zähler der gelesenen Nachricht
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich machen Sie für den DGET-Aufruf folgende Angaben:

KCOP

Im Feld KCOP tragen Sie den Operationscode DGET ein.

KCOM

Im Feld KCOM

- FT zum Verarbeiten der ersten Teilnachricht der ersten/neuen Nachricht
- NT zum Verarbeiten einer weiteren Teilnachricht der ersten/neuen Nachricht
- BF zum Lesen der ersten Teilnachricht einer Nachricht (Browsen ohne Löschen)
- BN zum Lesen einer weiteren Teilnachricht der Nachricht (Browsen ohne Löschen)

- PF zum Verarbeiten der ersten Teilnachricht einer bestimmten Nachricht
- PN zum Verarbeiten einer weiteren Teilnachricht einer bestimmten Nachricht

KCLA

Im Feld KCLA geben Sie die Länge des Nachrichtenbereichs an, in den die Nachricht gelesen werden soll. openUTM trägt die Länge der tatsächlich gelesenen Teilnachricht im Rückgabefeld KCRLM ein.

Wenn die Nachricht inclusive aller Teilnachrichten nicht gelesen werden soll, können Sie hier bei DGET FT den Wert 0 angeben. Ein nachfolgender DGET NT wird dann mit dem Returncode 10Z abgewiesen.

KCMF/kcfn
KCGTM

Das Feld **KCGTM** bzw. **KCMF/kcfn** muss wie folgt versorgt werden:

- mit Leerzeichen bei KCOM = FT/NT.
- mit Leerzeichen bei KCOM = BF, wenn der erste Teil der ersten Nachricht der Queue gelesen werden soll.
- mit der Erzeugungszeit der Nachricht bei KCOM = BN/PF/PN oder wenn bei KCOM = BF die nächste Nachricht gelesen werden soll. Die Erzeugungszeit wird beim letzten DGET BF im Feld KCRGTM zurückgegeben.

KCRN

Im Feld KCRN geben Sie den Namen der Queue an, aus der die Nachricht gelesen werden soll.

KCQTYP

Im Feld KCQTYP geben Sie den Typ der Queue an:

- T für TAC-Queue
- U für USER-Queue
- Q für Temporäre Queue

KCWTIME

Im Feld KCWTIME geben Sie bei DGET FT/BF an, wieviele Sekunden maximal auf das Eintreffen einer Nachricht gewartet werden soll. Es wird aber höchstens so lange gewartet wie bei der Generierung festgelegt wurde (MAX QTIME=). Die Angabe 0 bedeutet, dass nicht gewartet werden soll. Wird eine Wartezeit angegeben, muss das Folgeteilprogramm beim PEND einer TAC-Klasse zugeordnet sein.

Bei DGET NT/BN/PF/PN müssen Sie 0 angeben.

KCQRC

Im Feld KCQRC bestimmen Sie bei DGET BF das Verhalten nach Verarbeiten und Rücksetzen einer Transaktion. Sie können angeben:

- entweder den Wert, der beim vorhergehenden DGET BF-Aufruf im Feld KCRQRC zurückgegeben wurde. Damit wird sichergestellt, dass immer alle Nachrichten der Queue gelesen werden, eventuell wird eine verarbeitete Nachricht nach dem Rücksetzen noch einmal gelesen.
- oder den Wert -1 bzw. die Konstante KDCS_NO_QRC. Damit werden eventuell nicht alle Nachrichten der Queue gelesen.

Bei DGET FT/NT/BN/PF/PN müssen Sie 0 angeben.

KCDPID

Im Feld KCDPID geben Sie an:

- Binär null bei KCOM = FT/NT
- Leerzeichen, wenn bei KCOM = BF der erste Teil der ersten Nachricht gelesen werden soll.
- Die DPUT-ID bei KCOM = PF/PN oder wenn bei KCOM = BF/BN die nächste Nachricht/Teilnachricht gelesen werden soll. Die DPUT-ID wird beim vorhergehenden DGET BF im Feld KCRDPID zurückgegeben.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   die Adresse des KDCS-Parameterbereichs.

2. Parameter

   die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die Teilnachricht in der tatsächlichen, höchstens aber in der Länge des Nachrichtenbereichs.

KCRLM

im Feld KCRLM die tatsächliche Länge der gelesenen Teilnachricht, sofern in KCLA ein Wert > 0 angegeben wurde.

bei KCOM = FT/NT:

KCRMF/kcrfn

im Feld KCRMF den Namen der abstrakten Syntax der gelesenen Teilnachricht, wenn die Nachricht von einem OSI TP-Partner stammt. Sonst Leerzeichen.

KCRWVG

im Feld KCRWVG die Anzahl der Vorgänge, die bereits auf Nachrichten aus der angegebenen Queue warten (der aktuelle Vorgang wird dabei nicht mitgezählt). KCRWVG wird nur bei DGET FT versorgt.

KCRUS

im Feld KCRUS die UTM-Benutzerkennung, unter der die DGET-Nachricht erzeugt wurde. KCRUS wird nur bei DGET FT versorgt.

bei KCOM = BF:

KCRQRC

im Feld KCRQRC den Queue-spezifischen Redelivery-Zähler. Dieser Wert wird benötigt, um beim nächsten DGET BF-Aufruf das Feld KCQRC zu versorgen.

KCRGTM

im Feld KCRGTM die Erzeugungszeit der gelesenen Nachricht (binär). Dieser Wert wird benötigt, um beim nächsten DGET BF/BN/PF/PN-Aufruf das Feld KCGTM zu versorgen.

KCRDPID

im Feld KCRDPID die DPUT-ID der gelesenen Nachricht. Dieser Wert wird benötigt, um beim nächsten DGET BF/BN/PF/PN-Aufruf das Feld KCDPID zu versorgen.

bei KCOM = BF/BN/PF:

KCRRC

im Feld KCRRC den Redelivery-Zähler der gelesenen Nachricht. Dieser gibt an, wie oft die Nachricht nach Verarbeiten und Rücksetzen der Transaktion erneut zugestellt wurde. KCRRC kann maximal den Wert 254 erreichen. Ist dieser Wert erreicht und ist die Anzahl der erneuten Zustellungen nicht per Generierung beschränkt, dann wird nach jedem weiteren DGET BF/BN/PF immer der Wert 254 geliefert.

Bei allen Varianten:

KCRCCC

im Feld **KCRCCC** den KDCS-Returncode (siehe nächste Seite).

KCRCDC

im Feld **KCRCDC** den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim DGET-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde durchgeführt.
01Z	Längenkonflikt: KCLA < KCRLM, die Nachricht wurde abgeschnitten, weil die Teilnachricht länger ist als der Nachrichtenbereich.
04Z	Beim vorherigen DGET-Aufruf wurden noch nicht alle Nachrichtenteile gelesen; durch den aktuellen Aufruf DGET FT/BF/PF gehen die noch nicht gelesenen Nachrichtenteile verloren.

08Z Beim Lesen mit Warten (KCWTIME>0): Es liegt zurzeit keine Nachricht vor.

In diesem Fall sind keine weiteren DGET-Aufrufe erlaubt und das Teilprogramm muss mit PEND PA/PR beendet werden oder mit PGWT PR einen Wartepunkt setzen. Sobald eine Nachricht eintrifft oder die maximale Wartezeit abläuft oder die Queue gelöscht wird, setzt openUTM den Vorgang mit dem Folgeteilprogramm bzw. hinter dem PGWT PR fort. Ein Folgeteilprogramm muss einer TAC-Klasse zugeordnet sein.

10Z Bei DGET NT/BN/PN: Alle Teilnachrichten der Nachricht wurden bereits gelesen.

11Z Beim Lesen ohne Warten (KCWTIME = 0): Es liegt keine Nachricht vor.

40Z Bei DGET NT/BN/PN: Name oder Typ der angegebenen Queue passen nicht zum vorherigen DGET-Aufruf des aktuellen Teilprogrammlaufs. Dabei sind evtl. Teilnachrichten verloren gegangen. Es wurde keine Nachricht gelesen.

Bei DGET FT/PF: Es liegt ein Generierungsfehler vor (der Wert in MAX ...,RECBUF=... ist zu klein).

41Z Der DGET-Aufruf erfolgt unerlaubterweise im ersten Teil des Anmelde-Vorgangs oder der vorherige DGET-Aufruf ergab bereits den Returncode 08Z, d.h. es muss gewartet werden und es sind keine weiteren DGET-Aufrufe in diesem Teilprogramm erlaubt.

42Z Mögliche Ursachen:

- Der Wert in KCOM ist ungültig oder passt nicht zum vorherigen DGET-Aufruf (z.B. DGET PN nach DGET BF)
- oder der erste Nachrichtenteil wurde nicht in diesem Teilprogrammlauf gelesen
- oder es wurde inzwischen ein PGWT-Aufruf durchgeführt

43Z Der Wert in KCLA oder KCWTIME ist negativ bzw. ungültig oder bei DGET NT/BN/PF/PN wurde KCWTIME nicht mit 0 versorgt.

44Z Der Wert in KCRN ist ungültig, d.h.

- es gibt keine Queue mit dem angegebenen Namen und Typ oder
- die Queue wurde gelöscht oder
- der USER, der den Vorgang gestartet hat, bzw. dessen LTERM hat keine Berechtigung (KSET) zum Lesen der Queue oder
- der angegebene TAC ist nicht mit TYPE=Q generiert oder
- es wurde versucht, die Dead Letter Queue (KDCDLETQ) anders als mit Browse (Lesen ohne Löschen, d.h. DGET BF/BN) zu lesen.

45Z Der Wert in KCMF/KCGTM ist ungültig:

- Bei DGET FT/NT: KCMF wurde nicht mit Leerzeichen versorgt.
- Bei DGET BN/PF: Es existiert keine Nachricht mit der angegebenen DPUT-ID und Erzeugungszeit oder diese Nachricht wurde inzwischen gelöscht.

47Z Der Nachrichtenbereich fehlt oder die angegebene Bereichsadresse ist ungültig.

49Z Nicht verwendete Felder haben einen Wert ungleich binär null.

53Z Bei DGET BF/PF: Der Wert in KCDPID ist keine gültige DPUT-ID oder er passt nicht zu den Angaben in KCRN und KCQTYP.

Bei DGET BN/PN: Der Wert in KCDPID bzw. KCGTM stimmt nicht mit dem entsprechenden Wert des letzten DGET BF/PF überein.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z INIT missing in this program.

Eigenschaften des DGET-Aufrufs

- Nachrichtenlänge

  Die tatsächliche Nachrichtenlänge wird im Feld KCRLM zurückgegeben. Es gilt:

  - Bei KCRLM <= KCLA werden nur KCRLM Zeichen (Bytes) in den Nachrichtenbereich übertragen. Der Inhalt des restlichen Nachrichtenbereichs ist undefiniert.
  - Bei KCRLM > KCLA werden nur KCLA Zeichen in den Nachrichtenbereich übertragen. Der Rest (KCRLM - KCLA) geht verloren. Er kann mit einem nachfolgenden DGET nicht mehr gelesen werden.

  Bei der Beschreibung des MGET-Aufrufs finden Sie ein Beispiel, das das Verhalten von openUTM bei Längenkonflikten erläutert.
- Browsen und Verarbeiten
  - Beim Browsen (DGET BF/BN) können Nachrichten parallel von mehreren Vorgängen gelesen werden.
  - Beim Verarbeiten von Nachrichten (DGET FT/NT/PF/PN) kann jede Teilnachricht nur einmal gelesen werden. Sobald die Transaktion beendet wird, werden alle Teilnachrichten gelöscht.
  - Soll eine Nachricht nach dem Lesen (DGET BF/BN) mit DGET PF verarbeitet werden, dann muss die bei DGET BF zurückgegebene DPUT-ID und Erzeugungszeit beim DGET PF angegeben werden.
- Warten auf Nachrichten

  Innerhalb eines Dialog- oder Asynchron-Vorgangs dürfen solange DGET-Aufrufe für unterschiedliche Queues gegeben werden, bis auf eine Nachricht gewartet werden muss, d.h. liegt beim DGET-Aufruf mit Warten keine Nachricht vor, wird dies im Programm angezeigt durch KCRCCC = 08Z (KCWTIME > 0).

  Abhängig davon, ob auf eine Nachricht gewartet werden soll, sind folgende KDCS-Aufrufe zu programmieren:

  - Soll auf eine Nachricht gewartet werden, dann muss entweder das Teilprogramm mit PEND PA/PR beendet werden, damit außerhalb des Programmkontextes auf das Eintreffen der Nachricht gewartet werden kann, oder es muss mit PGWT PR ein Wartepunkt im Teilprogramm gesetzt werden.
  - Soll nicht auf eine Nachricht gewartet werden, dann muss entweder die Transaktion zurückgesetzt werden (RSET, PEND RS oder PGWT RB) oder der Vorgang muss mit PEND ER/FR beendet werden.

  Ansonsten wird beim PEND-Aufruf der Fehlercode 72Z zurückgegeben.

- Fortsetzen des Programms

  Wird auf eine DGET-Nachricht gewartet, startet openUTM das Folgeteilprogramm oder setzt das Programm hinter dem PGWT PR fort, sobald:

  1. eine neue Nachricht eintrifft oder
  2. die maximale Wartezeit abläuft oder
  3. die Queue gelöscht wird oder
  4. die verarbeitete Nachricht erneut zugestellt wird (Redelivery nach Rücksetzen der Transaktion), vorausgesetzt, es wird mit DGET FT gewartet oder es wurde KCQRC mit dem Wert des KB-Rückgabefeldes KCRQRC vom vorherigen Browse-Aufruf versorgt.

  In den Fällen a) bis c) werden alle an dieser Queue auf Nachrichten wartenden Vorgänge fortgesetzt (nicht nur der erste wartende Vorgang).

  Bei Redelivery (Fall d)) werden nur die Vorgänge fortgesetzt, die erneut zugestellte Nachrichten lesen sollen. Damit wird erreicht, dass eintreffende oder erneut zugestellte Nachrichten von den Vorgängen parallel gelesen werden können.

  Wird das Programm in einem PEND PA/PR-Folgeteilprogramm fortgesetzt, dann muss es einer TAC-Klasse zugeordnet sein, d.h. diese Funktionalität setzt voraus, dass TAC-Klassen generiert wurden. Ist dies nicht der Fall, wird der PEND-Aufruf mit KCRCCC=74Z abgewiesen.

- Erneute Zustellung (Redelivery)

  Wird eine Transaktion zurückgesetzt, dann wird eine verarbeitete Nachricht wieder in die Queue gestellt und kann erneut gelesen werden. Voraussetzung ist, dass die Anwendung entsprechend generiert wurde und die dort festgelegte Anzahl wiederholter Zustellungen noch nicht erreicht ist. Näheres siehe openUTM-Handbuch „Anwendungen generieren“, Operand REDELIVERY in der MAX-Anweisung.

- Im Vorgang KDCMSGTC und im Anmelde-Vorgang KDCSGNTC darf der Aufruf DGET nur ohne Angabe einer Wartezeit verwendet werden.

- Wird die Hauptnachricht eines Auftrags-Komplexes gelesen, für die ein positiver bzw. negativer Quittungsauftrag definiert ist (nur möglich bei TAC-Queues), dann wird
  - der positive Quittungsauftrag gestartet, nachdem die den DGET-Aufruf enthaltende Transaktion erfolgreich abgeschlossen wurde,
  - der negative Quittungsauftrag gestartet, nachdem die den DGET-Aufruf enthaltende Transaktion zurückgesetzt wurde ohne dass die Hauptnachricht erneut zugestellt wurde, d.h. keine Redelivery.

- Wird mit DGET aus einer Queue gelesen, für die ein Zugriffsschutz besteht, so muss die Benutzerkennung, unter der der Vorgang läuft, und der LTERM-Partner die Zugriffsberechtigung (KSET) für die entsprechende Queue besitzen. Ein Benutzer darf aber immer auf seine eigene USER-Queue zugreifen.
  Bei Anwendungen ohne explizit generierte Benutzerkennungen kann für die USER-Queues kein Zugriffsschutz vergeben werden.

- Sicherung fehlerhafter Nachrichten in der Dead Letter Queue

  Nachrichten aus TAC-Queues können im Fehlerfall als letzte Rückfallstufe in der globalen Dead Letter Queue gesichert werden. Dazu muss die Queue mit DEAD-LETTER-Q=YES generiert werden. Dann wird eine verarbeitete Nachricht beim Rücksetzen der Transaktion in die Dead Letter Queue gestellt, wenn sie nicht erneut zugestellt werden kann (siehe Redelivery) und kein negativer Quittungsauftrag definiert wurde.

  Beim Sichern einer Nachricht in der Dead Letter Queue wird die Anzahl der erneuten Zustellungen dieser Nachricht (Redelivery) ggf. auf null zurückgesetzt.

7.7 DPUT Erzeugen zeitgesteuerter Asynchron-Nachrichten

Mit dem Aufruf DPUT (delayed free message PUT) können Sie:

- Nachrichten an USER-Queues, TAC-Queues oder Temporäre Queues senden. Die Nachrichten an TAC-Queues können auch zeitgesteuert sein.
- zeitgesteuerte Asynchron-Aufträge mit ihren (Teil-)Nachrichten in eine Message Queue eintragen (Ausgabeaufträge an LTERM-Partner, Hintergrund-Aufträge an lokale Asynchron-Vorgänge, Hintergrund-Aufträge an ferne Asynchron-Vorgänge, die zuvor mit einem APRO AM-Aufruf adressiert wurden)
- Benutzerinformationen zu diesen Aufträgen protokollieren
- im Rahmen eines Auftrags-Komplexes Quittungsaufträge mit zugehörigen Benutzerinformationen erzeugen
- nur auf BS2000-Systemen: Druckoptionen für RSO-Drucker übergeben (= RSO-Parameterliste)

Den gewünschten Zeitpunkt geben Sie dabei im KDCS-Parameterbereich an (siehe Tabelle auf "DPUT-Aufruf ohne Auftrags-Komplex"). Zum angegebenen Zeitpunkt wird der Auftrag in die Warteschlange der auszuführenden Aufträge übernommen. Die Ausführung erfolgt also nicht exakt zur angegebenen Zeit, sondern frühestens zu diesem Zeitpunkt.

Die mit DPUT erzeugten Teilnachrichten werden von openUTM gesammelt und beim nächsten PEND-Aufruf abgeschlossen. Die Teilnachrichten werden nach
Transaktionsende als **eine** Nachricht in die entsprechende Queue eingetragen. Ausnahme: Bei formatierten Teilnachrichten an Terminals bildet jede Teilnachricht eine eigenständige Nachricht.

Wegen der verschiedenen Funktionen wird der DPUT-Aufruf in zwei Formaten dargestellt:

- DPUT-Aufruf ohne Auftrags-Komplex
- DPUT-Aufruf innerhalb eines Auftrags-Komplexes

Im Folgenden wird das Format des DPUT-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Message Queuing" finden Sie in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

7.7.1 DPUT-Aufruf ohne Auftrags-Komplex

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die beiden folgenden Tabellen zeigen die verschiedenen Möglichkeiten und die Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCLM	KCRN	KCMF/ kcfn	KCDF
Ausgabeauftrag im Formatmodus	"DPUT"	"NT"/" NE"	Länge	LTERM-Name	Formatkennzeichen	Bildschirmfunktion
Unix, Linux- und Windows-Systeme: Ausgabeauftrag im Zeilenmodus	"DPUT"	"NT"/" NE"	Länge	LTERM-Name	Leereichen	—
BS2000-Systeme: Ausgabeauftrag im Zeilenmodus	"DPUT"	"NT"/" NE"	Länge	LTERM-Name	Leerzeichen /Editprofil	Bildschirmfunktion / binär null
Hintergrund-Auftrag an Asynchron-Programm der gleichen Anwendung	"DPUT"	"NT"/" NE"	Länge	TAC	—	—
Nachricht an Service-gesteuerte Queue	"DPUT"	"QT"/" QE"	Länge	Name der Queue	—	—
Auftrag an Transportsystem-Anwendung	"DPUT"	"NT"/" NE"	Länge	LTERM-Name der Anwendung	Leerzeichen	binär null
Hintergrund-Auftrag an Auftragnehmer-Vorgang über LU6.1	"DPUT"	"NT"/" NE"	Länge	Vorgangs-Id	—	—
Hintergrund-Auftrag an Auftragnehmer-Vorgang über OSI TP	"DPUT"	"NT"/" NE"	Länge	Vorgangs-Id	Leerzeichen / Name der abstrakten Syntax	—
Benutzerinformation protokollieren	"DPUT"	"NI"/" QI"	Länge	wie bei zugehörigem DPUT NT/NE oder DPUT QT/QE	Leerzeichen	binär null

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	**KCOM**	**KCLM**	**KCRN**	**KCMF/ kcfn**	**KCDF**
BS2000-Systeme: Parameterliste für RSO-Drucker übergeben	"DPUT"	"RP"	Länge	LTERM-Name	Leerzeichen	binär null

NT, QT: Teilnachricht zum Auftrag
NE, QE: letzte Teilnachricht oder Gesamtnachricht zum Auftrag

NI, QI: Benutzerinformation zu einer nachfolgenden Nachricht
RP: RSO Parameter (BS2000-Systeme)

Eintrag im Feld KCOM	Zusätzliche Einträge im KDCS-Parameterbereich (KCPUT/kc_dput)						
	KCMOD	**KCTAG /kcday**	**KCSTD/ kchour**	**KCMIN**	**KCSEK/ kcsec**	**KCQTYP**	**sonstige Felder**
"NT"/"NE"	"A"	001 - 365 /366	00 - 23	00 - 59	00 - 59	—	—
	"R"	000 - 364 /365				—	
	Leerzeichen	—	—			—	
"QT"/"QE"	"A"	001 - 365 /366	00 - 23	00 - 59	00 - 59	binär null	binär null
	"R"	000 - 364 /365					
	Leerzeichen	binär null	binär null			"U" / "Q" / binär null	
"NI"	"A"	001 - 365 /366	00 - 23	00 - 59	00 - 59	binär null	binär null
	"R"	000 - 364 /365					
	Leerzeichen	binär null	binär null			"U" / "Q" / binär null	

Eintrag im Feld KCOM	Zusätzliche Einträge im KDCS-Parameterbereich (KCPUT/kc_dput)						
	KCMOD	**KCTAG /kcday**	**KCSTD/ kchour**	**KCMIN**	**KCSEK/ kcsec**	**KCQTYP**	**sonstige Felder**
"QI"	"A"	001 - 365 /366	00 - 23	00 - 59	00 - 59	binär null	binär null
	"R"	000 - 364 /365					
	Leerzeichen	binär null	binär null			"U" / "Q" / binär null	
"RP" (BS2000-Systeme)	"A"	001 - 365 /366	00 - 23	00 - 59	00 - 59	binär null	binär null
	"R"	000 - 364 /365					
	Leerzeichen	binär null					

A: absolute Zeitangabe
R: relative Zeitangabe (=Zeitabstand)

Bei DPUT NT/NE bzw. DPUT QT/QE müssen Sie die gleichen Zeitangaben machen wie beim zugehörigen DPUT NI bzw. DPUT QI.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, aus dem openUTM die Nachricht oder die Benutzerinformation oder die RSO-Parameterliste lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"DPUT"
KCOM	"NT"/"NE"/"QT"/"QE"/"NI"/"QI"/"RP"
KCLM	Länge in Byte
KCRN	LTERM-Name/TAC/Vorgangs-Id/Queue-Name
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax/ Editprofil (nur BS2000-Systeme)
KCDF	Bildschirmfunktion / binär null
KCMOD	"R"/"A"/'BLANK'
KCTAG/...	Zeitangabe (rel./abs.)
• KCTAG/kcday	• Tag (rel./abs.) / binär null
• KCSTD/kchour	• Stunde (rel./abs.) / binär null
• KCMIN	• Minute (rel./abs.) / binär null
• KCSEK/kcsec	• Sekunde (rel./abs.) / binär null
KCQTYP	"U" / "Q" / binär null
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_DPUTNT/KDCS_DPUTNE	(nb,kclm,kcrn,kcfn,kcdf,kcmod,kcday,kchour,kcmin,kcsec)
KDCS_DPUTQT/KDCS_DPUTQE	(nb,kclm,kcrn,kcfn,kcdf,kcmod,kcday,kchour,kcmin,kcsec,kcqtyp)
KDCS_DPUTNI	(nb,kclm,kcrn,kcmod,kcday,kchour,kcmin,kcsec)
KDCS_DPUTQI	(nb,kclm,kcrn,kcmod,kcday,kchour,kcmin,kcsec,kcqtyp)
KDCS_DPUTRP	(nb,kclm,kcrn,kcmod,kcday,kchour,kcmin,kcsec,)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich machen Sie für den DPUT-Aufruf folgende Einträge:

KCOP

Im Feld KCOP geben Sie den Operationscode DPUT an.

KCOM

Im Feld KCOM tragen Sie die gewünschte Operationsmodifikation ein:

- NT für Teilnachricht des Auftrags,
- NE für Gesamtnachricht bzw. letzte Teilnachricht des Auftrags,
- NI für Benutzerinformation zum Auftrag,
- QT für Teilnachricht an eine Service-gesteuerte Queue,
- QE für Gesamtnachricht bzw. letzte Teilnachricht an eine Service-gesteuerte Queue,
- QI für Benutzerinformation zur Nachricht an eine Service-gesteuerte Queue.
- RP für die Parameterliste eines RSO-Druckers (nur auf BS2000-Systemen).

KCLM

Im Feld KCLM geben Sie die Länge der Nachricht an, die gesendet werden soll (Länge Null darf auch angegeben werden).

Bei KCOM = RP ist dies die Länge der RSO-Parameterliste (nur auf BS2000-Systemen).

KCRN

Im Feld KCRN tragen Sie das Ziel der Nachricht ein:

- den Namen des LTERM-Partners, wenn dieser DPUT-Aufruf einen Ausgabeauftrag erzeugt oder eine RSO-Parameterliste übergibt.

- den Namen der USER-Queue, TAC-Queue oder Temporären Queue, wenn dieser DPUT-Aufruf eine Nachricht an eine Service-gesteuerte Queue erzeugt (KCOM=QT/QE/QI).
- den Transaktionscode eines Asynchron-Programms, wenn dieser DPUT-Aufruf einen Hintergrund-Auftrag erzeugt (ohne verteilte Verarbeitung),
- die Vorgangs-Id eines Auftragnehmer-Vorgangs, wenn dieser Hintergrund-Auftrag an einen Auftragnehmer-Vorgang gerichtet ist.

KCMF/kcfn

im Feld KCMF/kcfn:

- Leerzeichen im Zeilenmodus oder bei einem Auftrag an eine andere Anwendung ohne verteilte Verarbeitung
- bei Nachrichten an OSI TP-Partner:
  Name der abstrakten Syntax der Nachricht. Leerzeichen stehen dabei für die abstrakte Syntax von UDT; in diesem Fall wird als Transfer Syntax BER verwendet, und die Encodierung der Nachricht übernimmt openUTM.
  Wird hier ein Wert ungleich Leerzeichen angegeben, dann muss die Nachricht an openUTM in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax, übergeben werden.

Nur auf BS2000-Systemen:

- ein Formatkennzeichen im Formatmodus

  Bei Nachrichten an RSO-Drucker:

  Wenn ein Format speziell für RSO-Drucker erstellt wurde, muss das Formatierungssystem FHS den Druckertyp nicht kennen, da FHS eine logische Nachricht erzeugt, die von RSO in die physikalische Nachricht umgewandelt wird.

  Andernfalls muss FHS den Druckertyp, wie er bei RSO generiert ist, unterstützen, da es sonst zum Formatierungsfehler kommt.
- Leerzeichen beim Übergeben einer RSO-Parameterliste.
- ein Editprofil (bei Zeilenmodus oder einem RSO-Druckern). Ist die Nachricht an einen RSO-Drucker gerichtet, so wird nur der Parameter CCSNAME eines Editprofils ausgewertet. Der Zeichensatzname wird an RSO übergeben. Alle weiteren Parameter des Editprofils werden ignoriert, da es sich um VTSU-B Editoptionen handelt, die Nachricht aber von RSO aufbereitet wird.

Bei Nachrichten an einen Asynchron-Vorgang derselben Anwendung, an USER-, TAC- und Temporäre Queues oder an einen LU6.1-Partner ist dieses Feld irrelevant.

KCDF

Bei Ausgabe-Aufträgen an Terminals geben Sie im Feld KCDF die Bildschirmfunktion an. Bei Benutzerinformationen (KCOM = NI/QI), RSO-Parameterlisten und Aufträgen an Transportsystem-Anwendungen muss hier binär null angegeben werden.

Bei Hintergrundaufträgen, Nachrichten an LU6.1-Partner oder Nachrichten an USER-, TAC- und Temporäre Queues ist dieses Feld irrelevant.

Auf BS2000-Systemen muss binär null auch dann angegeben werden, wenn in KCMF/kcfn ein Editprofil oder ein #Format eingetragen ist.

KCMOD

Im Feld KCMOD wählen Sie die Art der Zeitangabe

- A für absolut
- R für relativ
- Leerzeichen, wenn der Auftrag ohne Wartezeit ausgeführt werden soll. Nachrichten an USER- und Temporäre Queues können nicht zeitgesteuert verschickt werden, daher muss in KCMOD Leerzeichen angegeben werden

KCTAG / ...

Hier machen Sie die notwendigen Zeitangaben für den Aufruf, absolut oder relativ entsprechend der Angabe in KCMOD, und zwar:

- bei absoluter Zeitangabe: im Feld KCTAG/kcday den Tag im Jahr (Industrietag), in KCSTD/kchour die Stunde, in KCMIN die Minute und in KCSEK/kcsec die Sekunde der gewünschten Uhrzeit.
- bei relativer Zeitangabe: im Feld KCTAG/kcday die Anzahl der Tage, in KCSTD/kchour die Anzahl der Stunden, in KCMIN die Anzahl der Minuten und in KCSEK/kcsec die Anzahl der Sekunden bis zum gewünschten Ausführungszeitpunkt.
- Bei KCMOD = Leerzeichen:
  binär null, wenn mit KCOM = NI/QI eine Benutzerinformation protokolliert werden soll oder wenn eine Nachricht an eine USER- oder Temporäre Queue geschickt werden soll (KCOM= QE/QT) (sonst irrelevant).

KCQTYP

Im Feld KCQTYP geben Sie bei Nachrichten an eine Queue den Typ der Queue an(nur in Verbindung mit KCOM=QT/QE/QI):

- Q für eine Temporäre Queue, die mit QCRE erzeugt wurde
- U für eine einer Benutzerkennung zugeordnete Queue (USER-Queue)
- binär null in allen anderen Fällen

Nachrichtenbereich

Tragen Sie die Nachricht oder Benutzerinformation ein, die Sie ausgeben oder die RSO-Parameterliste, die Sie übergeben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht, Benutzerinformation oder RSO-Parameterliste lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLM die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode,

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim DPUT-Aufruf ohne Auftrags-Komplex

Im Programm sind auswertbar:

000 Die Funktion wurde ausgeführt.

06Z Die Zeitangabe wechselt, ohne dass vorher DPUT NE gegeben wurde, d.h. mindestens eins der Felder KCMOD, KCTAG/kcday, KCSTD/kchour, KCMIN oder KCSEK/kcsec hat einen anderen Wert als beim ersten Nachrichtenteil (bei KCMOD=A/R). openUTM nimmt die Zeitangabe aus dem ersten DPUT-Aufruf und setzt die Nachricht fort.

40Z openUTM kann die Funktion nicht durchführen, siehe Eintrag in KCRCDC.

Mögliche Ursachen:

- KCDF enthält nicht binär null, obwohl dies in der speziellen Situation erforderlich wäre.
- Bei Aufträgen ohne verteilte Verarbeitung wechselt der Name in KCRN bzw. der Typ in KCQTYP, ohne dass der vorherige DPUT-Auftrag abgeschlossen wurde.
- Bei verteilter Verarbeitung: es existiert keine logische Verbindung zur Partner-Anwendung und KCMOD = "'BLANK'".

41Z Der Aufruf ist an dieser Stelle nicht erlaubt:

- der Aufruf wurde im ersten Teil des Anmelde-Vorgangs abgesetzt oder
- der Aufruf wurde im Anmelde-Vorgang nach einem SIGNON Aufruf und vor dem PEND PS Aufruf abgesetzt.

42Z Der Eintrag in KCOM ist ungültig.

43Z Die Längenangabe in KCLM ist negativ bzw. ungültig.

44Z Der Wert in KCRN oder KCQTYP ist ungültig. Mögliche Ursachen:

- Der Wert ist kein Transaktionscode eines Asynchron-Programms bzw. einer TAC-Queue und kein Name eines LTERM-Partners bzw. der LTERM-Name eines UPIC- oder HTTP-Clients, und auch keine gültige Vorgangs-Id, s. KCRCDC.
- Der Wert ist zwar der Transaktionscode eines Asynchron-Programms, aber der Transaktionscode ist gesperrt oder zugriffsgeschützt.
- KCQTYP=U: Der Wert in KCRN bezeichnet keinen Benutzer oder einen zugriffsgeschützten Benutzer.
- KCQTYP=Q: Der Wert in KCRN bezeichnet keine Temporäre Queue.
- Für die Dead Letter Queue (KDCDLETQ) können keine Asynchron-Nachrichten erzeugt werden.
- Bei KCOM = RP (nur auf BS2000-Systemen): Der Wert ist kein RSO-Drucker oder die aktuelle RSO-Version unterstützt diese Funktion nicht.

45Z Der Eintrag in KCMF/kcfn ist unzulässig.

Mögliche Ursachen:

- Das Formatkennzeichen in KCMF/kcfn ist nicht gültig.
- Ist die Nachricht an einen Partner gerichtet, mit dem über das OSI TP-Protokoll kommuniziert wird, dann bedeutet dieser Returncode, dass die im Feld KCMF/kcfn angegebene abstrakte Syntax für die Partner-Anwendung nicht generiert ist.

Nur auf BS2000-Systemen:

- Bei KCOM = RP: Kein Leerzeichen eingetragen
- Das Editprofil ist nicht generiert.
- Das Editprofil wechselt bei Nachrichtenteilen.

47Z Die Adresse des Nachrichtenbereichs ist ungültig.

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

51Z Nach einem DPUT NI/QI folgt kein DPUT NT/NE/QT/QE ans gleiche Ziel.

52Z Es wurde versucht, eine zeitgesteuerte Nachricht an eine USER-Queue oder Temporäre Queue zu schicken.

56Z Die Angabe in KCMOD ist ungültig oder die Zeitangabe in KCTAG/kcday, KCSTD/kchour, KCMIN oder KCSEK/kcsec ist ungültig oder liegt nicht innerhalb der generierten Zeitspanne.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Eigenschaften des DPUT-Aufrufs

- Der Nachrichtenbereich wird beim Ausführen des Aufrufs durch openUTM nicht verändert.
- In einem Teilprogramm können mehrere Aufträge erzeugt werden; die zugehörigen Nachrichten können aus mehreren Teilen bestehen.

- Bei Ausgabe von +Formaten, *Formaten oder Nachrichten im Zeilenmodus können Sie auch Bildasugabeschirmfunktionen verwenden, siehe Kapitel „Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme)“.
  Wenn Sie #Formate verwenden, müssen Sie KCDF mit binär null versorgen, sonst reagiert openUTM mit 40Z. Nur auf BS2000-Systemen: Auch bei der Verwendung von Editprofilen reagiert openUTM mit 40Z, falls KCDF nicht mit binär null versorgt ist.
- Bei einem PEND ER/FR, PEND RS oder RSET werden die mit DPUT erzeugten Aufträge verworfen.
- Bei DPUT-Aufrufen an eine andere UTM-Anwendung, die als Transportsystem-Anwendung generiert ist, muss der TAC jeweils am Anfang des Nachrichtenbereichs stehen.
- Die Nachricht wird gegebenenfalls formatiert, bevor sie ausgegeben wird.
- Nachrichten werden aufbewahrt bis:
  - das angesprochene Teilprogramm oder die Druckerausgabe beendet ist oder
  - bei Aufträgen an ferne Asynchron-Vorgänge die Übertragung erfolgreich abgeschlossen ist oder
  - die Nachricht mit KDCOUT am Terminal gelesen und eine neue Eingabe gemacht wurde (außer KDCLAST-Kommando)
  - die Nachricht an eine Queue mit einem DGET-Aufruf gelesen und die entsprechende Transaktion erfolgreich beendet wurde.
- Aufträge mit Nachrichten der Länge 0

  Erzeugt man eine Nachricht der Länge 0 (eine so genannte "leere Nachricht"), so wird
  - ein Hintergrund-Auftrag ausgeführt, d.h. der Asynchron-Vorgang wird gestartet, ohne dass er eine Nachricht erhält,
  - bei einem Ausgabeauftrag im Formatmodus ein leeres Format ausgegeben,
  - ein Ausgabeauftrag an eine Transportsystem-Anwendung zwar angenommen, aber von openUTM zu einem späteren Zeitpunkt verworfen.
- Ausgabeaufträge, die für ein Terminal bestimmt sind, werden in die Terminal Message Queue eingehängt, und können vom Benutzer mit dem Kommando KDCOUT gelesen werden. Pro KDCOUT-Kommando wird genau eine Nachricht gelesen. Jede Nachricht kann nur einmal gelesen werden. Bei wiederholter Eingabe des KDCOUT-Kommandos wird die nächste Nachricht aus der Terminal Queue gelesen.

  Dass für ein Terminal Asynchron-Nachrichten vorliegen, wird dem Terminalbenutzer bei Transaktionsende durch eine Meldung in der Systemzeile mitgeteilt.

  Auf BS2000-Systemen kann diese Ankündigung unterdrückt werden, wenn bei der Konfiguration für den betreffenden LTERM-Partner ANNOAMSG=N angegeben wurde (Standardwert: ANNOAMSG=Y). Asynchron-Nachrichten werden dann sofort am Bildschirm angezeigt. Dadurch kann die Dialogführung gestört werden. Der Terminalbenutzer kann sich jedoch mit dem KDCDISP-Kommando den letzten Bildschirm wieder anzeigen lassen.
- Wechselwirkungen zwischen FPUT- und DPUT-Aufrufen gibt es nicht, d.h. an ein bestimmtes Ziel können unabhängig voneinander DPUT-Aufrufe mit KCMOD = '"BLANK"' und FPUT-Aufrufe gesendet werden.
- Druckoptionen für RSO-Drucker (BS2000-Systeme)

  Wenn Sie Druckoptionen für Aufträge an RSO-Drucker verwenden, dann übergeben Sie zuerst mit DPUT RP die Liste mit den Druckoptionen, siehe RSO-Handbuch. Danach geben Sie mit DPUT NT/NE den eigentlichen Druckauftrag. Die Zeitangaben bei DPUT RP und DPUT NT/NE müssen übereinstimmen.
- Behandeln von Teilnachrichten

- Teilnachrichten im Zeilenmodus werden zusammengefasst und als **eine** Nachricht an den LTERM-Partner ausgegeben. Die mit DPUT erzeugten Teilnachrichten werden von openUTM gesammelt und beim nächsten PEND-Aufruf
  abgeschlossen, falls der Teilprogrammlauf sie noch nicht mit DPUT NE
  abgeschlossen hat. Die Teilnachrichten werden bei Transaktionsende als eine Nachricht an den LTERM-Partner oder an die andere Anwendung gesendet.
- Bei formatierten Teilnachrichten an Terminals erzeugt jede Teilnachricht ein neues Format. Der Formatname in KCMF/kcfn darf dabei wechseln. An einem Terminal muss jedes Format (jeder Teilnachricht) mit einem KDCOUT-Kommando abgeholt werden. Jeder DPUT NT-Aufruf erzeugt eine eigene Nachricht. Deshalb ist es nicht möglich, mit DPUT NT-Aufrufen einen Bildschirm aus mehreren Teilformaten aufzubauen. Die Formate treffen in der Reihenfolge ein, in der sie erzeugt wurden.
- Bei Teilnachrichten an Drucker ist ein Wechsel zwischen formatierten Teilnachrichten und nicht-formatierten Teilnachrichten (im Zeilen-Modus) möglich. Bei Teilnachrichten an Terminals führt dieser Wechsel zum Abschluss der alten und zum Beginn einer neuen Nachricht.
- Bei DPUT NT-Aufrufen an einen lokalen oder fernen Asynchron-Vorgang muss jede Teilnachricht mit einem eigenen FGET gelesen werden.
- Bei DPUT QT-Aufrufen an eine Service-gesteuerte Queue muss jede Teilnachricht mit einem eigenen DGET gelesen werden.
- Beim PEND wird die zuletzt mit DPUT aufgebaute Teilnachricht grundsätzlich als letzte Teilnachricht angenommen, auch wenn sie mit NT/QT ausgegeben wurde.
- Ein Wechsel des in KCRN angegebenen Ziels bei einer Folge von Teilnachrichten, ohne dass vorher ein DPUT NE/QE abgesetzt wurde, ist nur in bestimmten Fällen zulässig (siehe folgenden Punkt).
- Nur auf BS2000-Systemen: Wechselt das Editprofil innerhalb einer Folge von Teilnachrichten, die an ein Terminal gerichtet sind, dann reagiert openUTM mit 45Z.

- Die maximale Anzahl der DPUT NE/QE-Aufrufe in einer Transaktion wird vom Generierungsparameter RECBUF der KDCDEF-Anweisung MAX begrenzt. Pro DPUT NE werden 30 Bytes in diesem Puffer belegt. Ist der Puffer voll, so wird der DPUT NE mit KCRCDC=K704 abgewiesen.
- Parallele Nachrichten

  Parallele Nachrichten (d.h. Wechsel des Ziels vor DPUT NE/QE) sind immer dann erlaubt, wenn die Ziele unterschiedlichen Kategorien angehören. Dabei unterscheidet man die drei Kategorien:

  - LTERM-Partner, lokale Asynchron-Vorgänge und Service-gesteuerte Queues (KCRN=LTERM/TAC/Queue-Name)
  - Auftrags-Komplexe (KCRN = Komplex-Id)
  - ferne Asynchron-Vorgänge (KCRN = Vorgangs-Id) bzw. ferne TAC-Queues

  Innerhalb dieser Kategorien sind parallele Asynchron-Aufträge nur bei Aufträgen an ferne Asynchron-Vorgänge erlaubt.
  Ansonsten gilt: parallele Auftrags-Komplexe werden nicht unterstützt und der Wechsel des LTERM-/TAC-/Queue-Namens erfordert den Abschluss der Nachricht durch DPUT NE/QE.

- Einfluss von Generierungsparametern auf den DPUT-Aufruf

  Die folgenden Hinweise betreffen die Generierung der UTM-Anwendung. Näheres zu den einzelnen Generierungsparametern finden Sie im openUTM-Handbuch „Anwendungen generieren“.

  Die Schranken für die Zeitangabe im DPUT-Aufruf werden mit den Operanden DPUTLIMIT1 und DPUTLIMIT2 in der MAX-Anweisung festgelegt. Die gewünschte Ausführungszeit darf maximal um die Zeitangabe in DPUTLIMIT2 **vor** dem Zeitpunkt des DPUT-Aufrufs liegen und maximal um die Zeitangabe in DPUTLIMIT1 **nach** diesem Zeitpunkt:

  ```
  Aktuelle Zeit - DPUTLIMIT2 < Ausführungszeitpunkt < Aktuelle Zeit + DPUTLIMIT1
  ```

  Als aktuelle Zeit gilt der Zeitpunkt des DPUT-Aufrufs.

  Bei DPUT-Aufrufen mit KCMOD = A oder R an LTERM-Partner, die mit LTERM . . . , QAMSG=N generiert sind, prüft openUTM zum Zeitpunkt des DPUT-Aufrufs noch nicht, ob ein Client oder Drucker an den LTERM-Partner angeschlossen ist, sondern erst, wenn der im DPUT-Aufruf angegebene Zeitpunkt erreicht ist. Besteht dann keine Verbindung, so speichert openUTM die Nachricht so lange, bis eine Verbindung aufgebaut ist.

  Bei DPUT-Aufrufen mit KCMOD = "'BLANK"' an LTERM-Partner, die mit LTERM . . . , QAMSG=N generiert sind, prüft openUTM zum Zeitpunkt des DPUT-Aufrufs, ob an diesen LTERM-Partner ein Client/Drucker angeschlossen ist. Besteht keine Verbindung, so lehnt openUTM den Aufruf mit KCRCCC = 44Z, KCRCDC = K705 ab.

  Bei allen mit DPUT erzeugten Nachrichten an Terminals und Transportsystem-Anwendungen darf die Gesamtnachricht höchstens 32700 Bytes lang sein.

  Beim Aktualisieren der KDCFILE mit dem UTM-Tool KDCUPD kann in der TRANSFER-Anweisung differenziert festgelegt werden, welche Nachrichten in die neue KDCFILE übernommen werden sollen (siehe auch openUTM-Handbuch „Anwendungen generieren“).

- Mit dem Administrationsaufruf KDCINF STAT kann der UTM-Administrator die Anzahl der wartenden zeitgesteuerten Aufträge abfragen (siehe openUTM-Handbuch „Anwendungen administrieren“, Administrationsaufruf KDCINF).
- DPUT-Aufrufe bei verteilter Verarbeitung
  - Im Auftraggeber-Vorgang ist der KDCS-Parameterbereich vor dem DPUT-Aufruf genauso zu versorgen wie beim Senden von Nachrichten an eine TAC-Queue oder bei der Erzeugung von Hintergrund-Aufträgen an ein Asynchron-Programm der eigenen Anwendung. Lediglich als Rufname im Feld KCRN ist die Vorgangs-Id anzugeben, die vorher beim APRO AM-Aufruf für den Auftragnehmer-Vorgang vergeben wurde.
  - Nach Abschluss des Aufrufs DPUT NE/QE (letzter Nachrichtenteil oder Gesamtnachricht) oder nach dem KDCS-Returncode 40Z wird die Vorgangs-Id im Auftraggeber-Vorgang freigegeben. Die gleiche Identifikation kann dann für die Adressierung eines anderen Auftragnehmer-Vorgangs verwendet werden.
  - Der mit DPUT erzeugte Auftrag wird erst nach Ablauf der angegebenen Zeit an die Partner-Anwendung gesendet (falls dann eine freie Session bzw. Association verfügbar ist).
  - Parallele Asynchron-Aufträge an unterschiedliche Auftragnehmer-Vorgänge sind erlaubt.

- Benutzerinformationen (DPUT NI/QI)

  Eine Benutzerinformation gehört immer zu einem mit DPUT-Aufrufen erzeugten Auftrag. Die Benutzerinformation muss vor dem eigentlichen Auftrag erzeugt werden, wobei Adressat (Angabe in KCRN) und Zeitangabe bei Benutzerinformation und Auftrag übereinstimmen müssen. Eine Verletzung dieser Reihenfolge führt zu dem Fehler 51Z.
  Fehlt zu einer Benutzerinformation der zugehörige Auftrag, d.h. wird nur eine Benutzerinformation aufgebaut, bricht openUTM den Vorgang beim PEND-Aufruf mit KCRCCC = 86Z ab.

  Eine Benutzerinformation kann nur mit einem DADM-Aufruf gelesen werden; sie stellt eine Art "Protokollinformation" dar und wird nicht an den Adressaten übermittelt. Die Benutzerinformation eines Quittungsauftrags kann erst dann gelesen werden, wenn der Quittungsauftrag aktiviert wurde.

- Hintergrund-Aufträge an ein Asynchron-Programm der gleichen Anwendung

  Jeder Hintergrund-Auftrag bewirkt, dass zum angegebenen Zeitpunkt ein eigener Asynchron-Vorgang gestartet wird.

  Asynchron-Programme, die zeitgesteuert anlaufen, sollten überprüfen, ob ihre Arbeit noch sinnvoll ist oder ob sie sich sofort beenden. Die aktuelle Zeit und das aktuelle Datum sowie Zeit und Datum des Anwendungsstarts kann das Programm über den INFO-Aufruf erfahren.
  Benötigt das Programm den Zeitpunkt des DPUT-Aufrufs, so muss dieser in der Nachricht enthalten sein.

  Mit DPUT-Aufrufen können Sie auch periodisch wiederkehrende Asynchron-Aufträge realisieren. Dies geht mit einem Asynchron-Programm, das die periodisch auszuführende Aktion sowie einen DPUT-Aufruf an sich selbst enthält. Die Zeit kann man dabei relativ oder absolut angeben.

7.7.2 DPUT-Aufruf im Auftrags-Komplex

Mit dem DPUT-Aufruf innerhalb eines Auftrags-Komplexes können Sie:

- zeitgesteuerte Nachrichten, auch Basisaufträge genannt, mit ihren (Teil-)Nachrichten senden (Ausgabeaufträge an LTERM-Partner, an Asynchron-Vorgänge, Nachrichten an TAC-Queues oder Hintergrund-Aufträge an ferne Asynchron-Vorgänge bzw. TAC-Queues, die zuvor mit einem APRO AM-Aufruf adressiert wurden)
- Benutzerinformationen zu diesen Aufträgen protokollieren
- Quittungsaufträge mit zugehörigen Benutzerinformationen erzeugen

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die entsprechenden Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCLM	KCRN	KCMF/kcfn	KCDF
Basisauftrag an LTERM-Partner oder an lokalen Asynchron-Vorgang oder an lokale TAC-Queue	"DPUT"	"NT"/"NE"	Länge	Komplex-Id	Formatkennzeichen / Leerzeichen/ Name der abstrakten Syntax / Editprofile (nur BS2000-Systeme)	Bildschirmfunktion / binär null
Basisauftrag an TAC-Queue	"DPUT"	"QT"/"QE"	Länge	Komplex-Id	——	——
Basisauftrag an fernen Asynchron-Vorgang oder an ferne TAC-Queue	"DPUT"	"NT"/"NE"	Länge	Komplex-Id	——	——
Benutzerinformation zu einem Auftrag	"DPUT"	"NI"/"QI"	Länge	Komplex-Id	Leerzeichen	binär null
Quittungsauftrag erzeugen	"DPUT"	"+T"/"-T"	Länge	Komplex-Id	Leerzeichen	binär null
Benutzerinformation zum Quittungsauftrag	"DPUT"	"+I"/"-I"	Länge	Komplex-Id	Leerzeichen	binär null

Die Operationsmodifikationen im Feld KCOP haben folgende Bedeutungen:

NT/QT: Teilnachricht eines Basisauftrags

NE/QE: letzte Teilnachricht oder Gesamtnachricht eines Basisauftrags
NI/QI: Benutzerinformation zum Basisauftrag
+T/-T: positiver bzw. negativer Quittungsauftrag
+I/-I: Benutzerinformation zum positiven bzw. negativen Quittungsauftrag

Die Operationsmodifikationen QE/QT/QI beziehen sich auf einen Basisauftrag mit dem Ziel TAC-Queue. Die Angabe des Ziels USER- oder Temporäre Queue ist beim DPUT-Aufruf im Auftrags-Komplex nicht möglich, da KCQTYP hier nicht ausgewertet wird und das Ziel beim zugehörigen MCOM-Aufruf angegeben wird.

Die Zeitangaben bei DPUT NT/NE/NI bzw. DPUT QT/QE/QI werden wie beim DPUT-Aufruf ohne Auftrags-Komplex angegeben. Bei DPUT NI/QI/+T/-T/+I/-I müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, aus dem openUTM die Nachricht oder die Benutzerinformation lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"DPUT"
KCOM	"NT"/"NE"/"NI"/"QT"/"QE"/"QI"/"+T"/"-T"/ +I" / "-I"
KCLM	Länge in Byte
KCRN	Komplex-Identifikation
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax / Name des Editprofils (nur BS2000-Systemen)
KCDF	Bildschirmfunktion/binär null
KCMOD	"R"/"A"/'BLANK'/binär null
KCTAG / ...	
• KCTAG/kcday	Tag (rel./abs.)/'BLANK'/binär null
• KCSTD/kchour	Stunde (rel./abs.)/'BLANK'/binär null
• KCMIN	Minute (rel./abs.)/'BLANK'/binär null
• KCSEK/kcsec	Sekunde (rel./abs.)/'BLANK'/binär null
KCQTYP	irrelevant (wird nicht ausgewertet)
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_DPUTNT/KDCS_DPUTNE	(nb,kclm,kcrn,kcfn,kcdf,kcmod,kcday,kchour,kcmin, kcsec)
KDCS_DPUTQT/KDCS_DPUTQE	(nb,kclm,kcrn,kcfn,kcdf,kcmod,kcday,kchour,kcmin, kcsec,kcqtyp)
KDCS_DPUTNI	(nb,kclm,kcrn,kcmod,kcday,kchour,kcmin,kcsec)
KDCS_DPUTQI	(nb,kclm,kcrn,kcmod,kcday,kchour,kcmin,kcsec, kcqtyp)
KDCS_DPUTPT/KDCS_DPUTMT/KDCS_DPUTPI /KDCS_DPUTMI	(nb,kclm,kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich tragen Sie für den DPUT-Aufruf innerhalb von Auftrags-Komplexen folgende Angaben ein:

KCOP

Im Feld KCOP geben Sie den Operationscode DPUT an.

KCOM

Im Feld KCOM tragen Sie die gewünschte Operationsmodifikation ein:

- NT/QT für Teilnachricht des Basisauftrages.
- NE/QE für Gesamtnachricht bzw. letzte Teilnachricht des Basisauftrags.
- NI/QI für eine Benutzerinformation zum Basisauftrag.
- +T für einen positiven Quittungsauftrag
- -T für einen negativen Quittungsauftrag
- +I für eine Benutzerinformation zum positiven Quittungsauftrag
- -I für eine Benutzerinformation zum negativen Quittungsauftrag

Die Operationsmodifikationen QE/QT/QI beziehen sich auf einen Basisauftrag mit dem Ziel TAC-Queue.

KCLM

Im Feld KCLM geben Sie die Länge der Nachricht im Nachrichtenbereich an, die gesendet werden soll (Länge Null darf auch angegeben werden).

KCRN

Im Feld KCRN tragen Sie die Komplex-Identifikation (Komplex-Id) ein, die zum Auftrags-Komplex gehört (wird im MCOM-Aufruf vergeben).

KCMF/kcfn

Im Feld KCMF/kcfn (bei Nachrichten an einen Asynchron-Vorgang derselben Anwendung oder an einen LU6.1-Partner irrelevant) tragen Sie ein:

- Leerzeichen im Zeilenmodus oder bei einem Auftrag an eine andere Anwendung ohne verteilte Verarbeitung

bei Nachrichten an OSI TP-Partner:

- Name der abstrakten Syntax der Nachricht. Leerzeichen stehen dabei für die abstrakte Syntax von UDT; in diesem Fall wird als Transfer Syntax BER verwendet, und die Encodierung der Nachricht übernimmt openUTM.
  Wird hier ein Wert ungleich Leerzeichen angegeben, dann muss die Nachricht an openUTM in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax, übergeben werden.

Nur auf BS2000-Systemen:

- ein Formatkennzeichen im Formatmodus

  Bei Nachrichten an RSO-Drucker:

  Wenn ein Format speziell für RSO-Drucker erstellt wurde, muss das Formatierungssystem FHS den Druckertyp nicht kennen, da FHS eine logische Nachricht erzeugt, die von RSO in die physikalische Nachricht umgewandelt wird.

  Andernfalls muss FHS den Druckertyp, wie er bei RSO generiert ist, unterstützen, da es sonst zum Formatierungsfehler kommt.
- ein Editprofil (bei Zeilenmodus oder einem RSO-Drucker)
  Ist die Nachricht an einen RSO-Drucker gerichtet, so wird nur der Parameter CCSNAME eines Editprofils ausgewertet. Der Zeichensatzname wird an RSO übergeben. Alle weiteren Parameter des Editprofils werden ignoriert, da es sich um VTSU-B Editoptionen handelt, die Nachricht aber von RSO aufbereitet wird.

KCDF

Bei Ausgabe-Aufträgen an Terminals geben Sie im Feld KCDF die Bildschirmfunktion an. Bei Benutzerinformationen (KCOM = NI/QI) oder Aufträgen an Transportsystem-Anwendungen muss hier binär null angegeben werden.

Bei Hintergrundaufträgen und Nachrichten an TAC-Queues ist dieses Feld irrelevant.

Auf BS2000-Systemen muss binär null auch dann angegeben werden, wenn in KCMF/kcfn ein Editprofil oder ein #Format eingetragen ist.

KCMOD

Im Feld KCMOD wählen Sie bei Nachrichten an den Basisauftrag (KCOM = NT/NE/NI bzw. QT/QE/QI) die Art der Zeitangabe:

- A für absolut
- R für relativ
- Leerzeichen, wenn der Auftrag ohne Wartezeit ausgeführt werden soll

Bei Nachrichten an Quittungsaufträge (KCOM = +T/-T/+I/-I) muss binär null eingetragen werden.

KCTAG / ...

In diesen Feldern machen Sie bei KCOM = NT/NE/NI bzw. QT/QE/QI die notwendigen Zeitangaben wie beim DPUT-Aufruf ohne Auftrags-Komplex, bei KCOM = +T/-T/+I/-I tragen Sie binär null ein.

KCQTYP

Das Feld KCQTYP wird nicht ausgewertet.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Nachricht oder Benutzerinformation ein, die Sie ausgeben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

Die Adresse des KDCS-Parameterbereichs.

2. Parameter

Die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht (bzw. Benutzerinformation) lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLM die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim DPUT-Aufruf mit Auftrags-Komplex

Im Programm sind auswertbar:

000	Die Funktion wurde ausgeführt.

06Z Die Zeitangabe wechselt, ohne dass vorher DPUT NE gegeben wurde, d.h. mindestens eins der Felder KCMOD, KCTAG/kcday, KCSTD/kchour, KCMIN oder KCSEK/kcsec hat einen anderen Wert als beim ersten Nachrichtenteil (bei KCMOD=A/R). openUTM nimmt die Zeitangabe aus dem ersten DPUT-Aufruf und setzt die Nachricht fort.

40Z openUTM kann die Funktion nicht durchführen, siehe Eintrag in KCRCDC.

Bei verteilter Verarbeitung: KCMOD = '"BLANK"' und es existiert keine logische Verbindung zur Partner-Anwendung.

41Z Der Aufruf ist an dieser Stelle nicht erlaubt:

- Nach Abschluss eines Basisauftrags (DPUT NE/QE) soll ein weiterer Basisauftrag gegeben oder eine Benutzerinformation protokolliert werden oder
- das Formatkennzeichen wechselt bei mehreren DPUT NT oder
- der Aufruf wurde im ersten Teil des Anmelde-Vorgangs abgesetzt oder
- der Aufruf wurde im Anmelde-Vorgang nach einem SIGN ON Aufruf und vor dem PEND PS Aufruf abgesetzt.

42Z Der Eintrag in KCOM ist ungültig oder es wurde KCOM = +T/-T angegeben, ohne dass ein Auftrags-Komplex oder ein Ziel für den Quittungsauftrag definiert wurde.

43Z Die Längenangabe in KCLM ist negativ bzw. ungültig.

44Z Die in KCRN angegebene Komplex-Identifikation ist ungültig.

45Z Der Eintrag in KCMF/kcfn ist unzulässig. Mögliche Ursachen:

- Das Formatkennzeichen in KCMF/kcfn ist nicht gültig.
- Ist die Nachricht an einen Partner gerichtet, mit dem über das OSI TP-Protokoll kommuniziert wird, dann bedeutet dieser Returncode, dass die im Feld KCMF/kcfn angegebene abstrakte Syntax für die Partner-Anwendung nicht generiert ist.

Nur auf BS2000-Systemen:

- Das Editprofil ist nicht generiert.
- Das Editprofil wechselt bei Nachrichtenteilen an Terminals.

47Z Die Adresse des Nachrichtenbereichs ist ungültig.

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

51Z Die Reihenfolge der DPUT-Aufrufe wurde nicht eingehalten (siehe unten).

56Z Der Eintrag in KCMOD ist unzulässig oder die Zeitangabe in KCTAG/kcday, KCSTD/kchour, KCMIN oder KCSEK/kcsec ist ungültig oder liegt nicht innerhalb der generierten Zeitspanne.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Eigenschaften des DPUT-Aufrufs innerhalb von Auftrags-Komplexen

- Für die Reihenfolge der DPUT-Aufrufe gelten die folgenden Regeln:
  - Die Benutzerinformation muss vor dem zugehörigen (Quittungs-)Auftrag geschrieben werden: DPUT NI vor DPUT NT/NE bzw. DPUT QI vor DPUT QT/QE, DPUT +I vor DPUT +T und DPUT -I vor DPUT -T. Dabei ist nur ein DPUT +I/-I pro DPUT +T/-T erlaubt.
    Die Benutzerinformation eines Quittungsauftrags kann erst dann gelesen werden, wenn der Quittungsauftrag aktiviert wurde.
  - Vor dem ersten Quittungsauftrag muss der Basisauftrag begonnen werden, d.h. DPUT NT/NE bzw. DPUT QT /QE vor dem ersten DPUT +T/-T. Es sind mehrere DPUT +T oder DPUT -T erlaubt; diese werden jeweils als **ein** positiver
    Quittungsauftrag (bei +T) und als **ein** negativer Quittungsauftrag (bei -T) betrachtet.
- Ein Wechsel des Formatkennzeichens bei mehreren Nachrichtenteilen (DPUT NT) ist nicht erlaubt.
- Bei einem Returncode 40Z in einem DPUT-Aufruf gehen alle Nachrichten und Informationen, die zu dem Auftrags-Komplex gehören, verloren; die Komplex-Id wird freigegeben.

  Bei DPUT-Nachrichten über verteilte Verarbeitung wird bei 40Z auch die Vorgangs-Id freigegeben.

  Bei allen anderen Returncodes kleiner 70Z bleiben sämtliche Informationen des Komplexes einschließlich der Komplex-Id erhalten.
- Parallele Asynchron-Aufträge (d.h. Wechsel des Ziels vor DPUT NE/QE) sind erlaubt, wenn das Ziel einer anderen Kategorie angehört, d.h. vor DPUT NE ist ein DPUT mit KCRN = LTERM/TAC/Queue oder KCRN = VGID zulässig.

  Beispiel für parallele Asynchron-Aufträge

KDCS-Aufruf	Ziel und Identifikation	Bemerkungen
MCOM BC	KCRN = DRUCKER1 KCCOMID = *KOMPLEX KCPOS = ATAC5	Beginn eines Auftrags-Komplexes und Auftragsziele definieren
DPUT NT	KCRN = *KOMPLEX	1. Teilnachricht zum Basisauftrag
DPUT NE	KCRN = ATAC1	Hintergrund-Auftrag an ein Programm
APRO AM	KCRN = LTAC KCPA = APPL1 KCPI = > VGID	Auftragnehmer-Vorgang adressieren
DPUT NE	KCRN = > VGID	Hintergrund-Auftrag an Auftragnehmer-Vorgang
DPUT NE	KCRN = *KOMPLEX	2. Teilnachricht zum Basisauftrag
DPUT +T	KCRN = *KOMPLEX	Positiver Quittungsauftrag
MCOM EC	KCRN = binär null KCCOMID = *KOMPLEX	Ende des Auftrags-Komplexes

7.8 FGET Empfangen einer Asynchron-Nachricht

Mit dem Aufruf FGET (free message GET) wird eine Asynchron-Nachricht oder -Teilnachricht aus der dem Service zugeordneten Message Queue in den Nachrichtenbereich eingelesen.

Der FGET-Aufruf darf nur im ersten Teilprogrammlauf und Verarbeitungsschritt eines Asynchron-Vorgangs gegeben werden. Jede Nachricht kann nur einmal gelesen werden. Dabei ist jede Teilnachricht mit einem eigenen FGET zu lesen. Nur nach einem RSET-Aufruf kann eine (Teil-)Nachricht noch einmal gelesen werden.

Im Folgenden wird das Format des FGET-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Message Queuing" finden Sie in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**		
	KCOP	**KCLA**	**KCMF/kcfn**
Asynchron-Nachricht lesen	"FGET"	Länge	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax/ Editprofile (nur BS2000-Systeme)

Versorgen des 2. Parameters

Hier müssen Sie die Adresse des Nachrichtenbereichs bereitstellen, in den openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"FGET"
KCLA	Länge in Byte
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax/ Name des Editprofils (nur BS2000-Systeme)

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_FGET	(nb,kcla,kcfn)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Länge
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Formatkennzeichen/Leerzeichen
KCRRC	Redelivery-Zähler

Im KDCS-Parameterbereich machen Sie für den FGET-Aufruf folgende Angaben:

KCOP

Im Feld KCOP tragen Sie den Operationscode FGET ein.

KCLA

Im Feld KCLA geben Sie die Länge an, in der die Nachricht gelesen werden soll. Sie darf höchstens so groß sein wie der Nachrichtenbereich, in den die Nachricht gelesen werden soll. Länge Null bedeutet: kein Nachrichtenempfang. Eine eventuell vorhandene Nachricht geht verloren.

KCMF/kcfn

Im Feld KCMF/kcfn geben Sie das Format der zu lesenden Nachricht an:

- bei Zeilenmodus: Leerzeichen

  oder auf BS2000-Systemen: Name des **Editprofils** (wird beim INIT im Feld KCRMF/kcrfn zurückgegeben). Dieser Name beginnt mit einem Leerzeichen.
- im Formatmodus auf BS2000-Systemen: Formatkennzeichen des erwarteten (Teil) Formats. Dieses wurde beim INIT- bzw. beim vorangegangenen FGET-Aufruf im Feld KCRMF/kcrfn zurückgegeben.
- bei einer Nachricht von einem Teilprogramm derselben Anwendung oder von einem LU6.1-Partner: irrelevant.

bei Nachricht von einem OSI TP-Partner:

- Name der abstrakten Syntax der Nachricht.
  Dieser Name wurde im Feld KCRMF/kcrfn bei dem vorausgegangen INIT-Aufruf zurückgegeben.
  Leerzeichen stehen dabei für die abstrakte Syntax von UDT codiert nach BER (Basic Encoding Rules); nur in diesem Fall wird die Nachricht von openUTM an das Teilprogramm bereits decodiert übergeben.
  Wird hier ein Wert ungleich Leerzeichen angegeben, dann übergibt openUTM die Nachricht an das Teilprogramm in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax; das Teilprogramm muss die Umsetzung in die lokale Darstellung selbst übernehmen. Dies kann z.B. mit Hilfe eines ASN.1-Compilers geschehen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die (Teil-)Nachricht in der tatsächlichen, höchstens aber in der angeforderten Länge.

KCRLM

im Feld KCRLM die tatsächliche Länge der (Teil-)Nachricht, ggf. in Abweichung von der angeforderten Länge in KCLA des Parameterbereichs.

KCRCCC

im Feld KCRCCC den KDCS-Returncode (siehe nächste Seite).

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

im Feld KCRMF/kcrfn:

- nach dem Lesen im Zeilenmodus: Leerzeichen

  oder auf BS2000-Systemen: Name des Editprofils der letzten Ausgabenachricht.
- nach dem Lesen von einem Partner-Vorgang:
  Name des Formatkennzeichens bzw. der abstrakten Syntax der nächsten (Teil-)Nachricht.

Nur auf BS2000-Systemen:

- nach dem Lesen eines ganzen Formats: Kennzeichen des zuletzt eingelesenen Formats. Es ist immer gleich dem Kennzeichen des letzten Ausgabeformats.
- nach dem Lesen eines Teilformats: Kennzeichen des nächsten Teilformats mit Eingabedaten.
- nach dem Lesen des letzten Teilformats: Kennzeichen des zuletzt eingelesenen Teilformats. In diesem Fall ist KCRMF = KCMF (bzw.: kcrfn=kcfn).

KCRRC

im Feld KCRRC den Redelivery-Zähler der gelesenen Nachricht. Dieser enthält die Anzahl der erneuten Zustellungen der FGET-Nachricht nach abnormalem Beenden der Asynchron-Vorgänge in der ersten Transaktion.

Absenderangaben usw. stehen im KB-Kopf (von openUTM beim INIT eingetragen).

KDCS-Returncodes im Feld KCRCCC beim FGET-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde durchgeführt.

01Z Längenkonflikt: KCLA < KCRLM, die Nachricht wurde abgeschnitten.

03Z Bei Teilformaten:
KCMF/kcfn enthält nicht den Namen des nächsten Teilformats; der Nachrichtenbereich ist unverändert und KCRLM = 0.

Eingabenachricht von einem OSI TP-Partner:
KCMF/kcfn enthält nicht den Namen der abstrakten Syntax der als Nächstes zu lesenden Nachricht. Es wird keine Nachricht in den Nachrichtenbereich übertragen.

05Z Bei Einzelformaten war am Bildschirm ein anderes Format ausgegeben als in KCMF/kcfn eingetragen.

Nur auf BS2000-Systemen:
Im Zeilenmodus war am Bildschirm ein anderes Editprofil ausgegeben als in KCMF/kcfn eingetragen.

10Z Die Nachricht bzw. alle Teilnachrichten wurden bereits vollständig gelesen.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z Die Funktion wurde in einem Folge-Teilprogramm oder nach einem PGWT-Aufruf eines Asynchron-Vorgangs oder in einem Dialog-Vorgang aufgerufen bzw. im Teilprogrammlauf wurde noch kein INIT aufgerufen.

73Z Die Längenangabe in KCLA ist negativ oder ungültig.

77Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

Eigenschaften des FGET-Aufrufs

- Die tatsächliche Nachrichtenlänge wird im Feld KCRLM zurückgegeben. Es gilt:
  - Bei KCRLM <= KCLA werden nur KCRLM Zeichen (Bytes) in den Nachrichtenbereich übertragen. Der Inhalt des restlichen Nachrichtenbereichs ist undefiniert.
  - Bei KCRLM > KCLA werden nur KCLA Zeichen in den Nachrichtenbereich übertragen. Der Rest (KCRLM - KCLA) geht verloren. Er kann mit einem nachfolgenden FGET nicht mehr gelesen werden.

  Bei der Beschreibung des MGET-Aufrufs finden Sie ein Beispiel, das das Verhalten von openUTM bei Längenkonflikten erläutert.
- Ein Teilprogramm kann auch Asynchron-Nachrichten der Länge 0 empfangen, wenn z.B.
  - eine Funktionstaste eingegeben wurde, ohne eine Nachricht zuzuordnen,
  - ein Transaktionscode ohne weitere Daten gesendet wurde,
  - ein Programm der gleichen Anwendung einen Hintergrund-Auftrag mit einer Nachricht der Länge 0 gegeben hat.
- TAC-Eingabe von einem Terminal oder einer Transportsystem-Anwendung:

- Wird ein Asynchron-TAC im Zeilenmodus eingegeben und in KCMF/kcfn ein Formatkennzeichen eingetragen, so wird nicht - wie beim MGET - der Returncode KCRCCC = 05Z gesetzt, sondern KCRCCC = 000 (falls der Aufruf sonst fehlerfrei ist).
- Bei Eingaben aus Teilformaten muss jedes Teilformat mit einem eigenen FGET gelesen werden.
- Wird zusammen mit dem Asynchron-TAC eine Nachricht eingegeben, so wird der TAC von der Nachricht abgetrennt: Der TAC wird nicht in den Nachrichtenbereich eingelesen, sondern steht nach dem INIT im KB-Kopf zur Verfügung.
- openUTM nimmt keine Umsetzung von Kleinbuchstaben in Großbuchstaben vor.

  Auf BS2000-Systemen lässt sich eine Umsetzung jedoch über Editprofile erreichen.

- Die Anzahl der erneuten Zustellungen wird im Feld KCRRC zurückgegeben. Eine Nachricht wird immer dann erneut zugestellt, wenn ein Asynchron-Vorgang in der ersten Transaktion abnormal beendet wurde. Voraussetzung ist, dass die Anwendung entsprechend generiert wurde und die generierte maximale Anzahl der erneuten Zustellungen noch nicht erreicht ist. Näheres siehe openUTM-Handbuch „Anwendungen generieren“, Operand REDELIVERY in der MAX-Anweisung.
- Sicherung fehlerhafter Nachrichten in der Dead Letter Queue:

  Asynchron-Nachrichten zu Transaktionscodes können im Fehlerfall als letzte Rückfallstufe in der globalen Dead Letter Queue gesichert werden. Dazu muss der TAC mit DEAD-LETTER-Q=YES generiert werden. Dann wird die FGET-Nachricht bei abnormaler Beendigung des Asynchron-Vorgangs ohne erfolgreichen Abschluss einer Transaktion in die Dead Letter Queue gestellt, wenn sie nicht erneut zugestellt werden kann (siehe Redelivery) und kein negativer Quittungsauftrag definiert wurde. Sobald ein Asynchron-Vorgang einen Sicherungspunkt erreicht hat, ist sowohl Redelivery als auch Sicherung der FGET-Nachricht in der Dead Letter Queue ausgeschlossen, da die Nachricht dann als erfolgreich verarbeitet gilt.

  Beim Sichern einer Nachricht in der Dead Letter Queue wird die Anzahl der erneuten Zustellungen dieser Nachricht (Redelivery) ggf. auf Null zurückgesetzt.

> ¡ Jede mit FPUT NT gesendete Teilnachricht muss mit einem eigenen FGET gelesen werden.

7.9 FPUT Erzeugen von Asynchron-Nachrichten

Mit dem Aufruf FPUT (free message PUT) können Sie Nachrichten oder Nachrichtenteile für Message Queues erzeugen, die von openUTM jeweils in Empfänger-spezifische Queues eingetragen werden:

- Ausgabeaufträge an LTERM-Partner
- Hintergrund-Aufträge an lokale Asynchron-Vorgänge
- Hintergrund-Aufträge an ferne Asynchron-Vorgänge, die zuvor mit APRO AM-Aufrufen adressiert wurden
- Asynchron-Nachrichten an lokale oder entfernte TAC-Queues
- Nur auf BS2000-Systemen: Druckoptionen für RSO-Drucker übergeben (= RSO-Parameterliste)

Die mit FPUT erzeugten Teilnachrichten werden von openUTM gesammelt und beim nächsten PEND-Aufruf abgeschlossen. Die Teilnachrichten werden bei Transaktionsende als eine Nachricht in die entsprechende Message Queue eingetragen. Ausnahme: Bei formatierten Teilnachrichten an Terminals bildet jede Teilnachricht eine eigenständige Nachricht.

Im Falle von TAC-Queues muss der Empfänger die Nachricht in einer eigenen Transaktion aus der Queue lesen.

Nachrichten bleiben solange in einer Message Queue erhalten bis sie:

- erfolgreich gesendet (LTERM-Partner) oder
- erfolgreich verarbeitet (Asynchron-Vorgang) oder
- erfolgreich gelesen wurden (TAC-Queue).

Im Folgenden wird das Format des FPUT-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Message Queuing" finden Sie in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCLM	KCRN	KCMF/kcfn	KCDF
Ausgabeauftrag im Formatmodus	"FPUT"	"NT"/"NE"	Länge	LTERM-Name	Formatkennzeichen	Bildschirmfunktion
Unix-, Linux-, Windows-Systeme: Ausgabeauftrag im Zeilenmodus	"FPUT"	"NT"/"NE"	Länge	LTERM-Name	Leerzeichen	——
BS2000-Systeme: Ausgabeauftrag im Zeilenmodus	"FPUT"	"NT"/"NE"	Länge	LTERM-Name	Leerzeichen /Editprofil	Bildschirmfunktion / binär null
Nachricht an Asynchron-Programm oder an TAC-Queue der gleichen Anwendung	"FPUT"	"NT"/"NE"	Länge	TAC / Name einer TAC-Queue	——	——
Ausgabe-Auftrag an Transportsystem-Anwendung	"FPUT"	"NT"/"NE"	Länge	LTERM-Name der Anwendung	Leerzeichen	binär null
Hintergrund-Auftrag über LU6.1	"FPUT"	"NT"/"NE"	Länge	Vorgangs-Id	——	——
Hintergrund-Auftrag über OSI TP	"FPUT"	"NT"/"NE"	Länge	Vorgangs-Id	Name der abstrakten Syntax / Leerzeichen	——
BS2000-Systeme: Parameterliste für RSO-Drucker übergeben	"FPUT"	"RP"	Länge	LTERM-Name	Leerzeichen	binär null

NT: Teilnachricht zum Auftrag
NE: letzte Teilnachricht oder Gesamtnachricht zum Auftrag
RP: RSO Parameterliste (BS2000-Systeme)

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, aus dem openUTM die Nachricht oder die RSO-Parameterliste lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"FPUT"
KCOM	"NT"/"NE"/"RP"
KCLM	Länge in Byte
KCRN	LTERM-Name/TAC/TAC-Queue/Vorgangs-Id
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax / Editprofile (nur auf BS2000-Systemen)
KCDF	Bildschirmfunktion / binär null
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_FPUTNT / KDCS_FPUTNE	(nb,kclm,kcrn,kcfn,kcdf)
KDCS_FPUTRP	(nb,kclm,kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den FPUT-Aufruf ein:

KCOP

im Feld KCOP den Operationscode FPUT.

KCOM

im Feld KCOM entweder NT für Teilnachricht oder NE für Gesamtnachricht bzw. letzte Teilnachricht oder RP für eine RSO-Parameterliste.

KCLM

im Feld KCLM die Länge der Nachricht im Nachrichtenbereich, die gesendet werden soll (Länge Null darf auch angegeben werden).

Nur auf BS2000-Systemen: Bei KCOM = RP ist dies die Länge der Datenstruktur für die RSO-Parameterliste.

KCRN

im Feld KCRN abhängig vom Empfänger der Nachricht:

- den Namen eines LTERM-Partners, wenn dieser FPUT-Aufruf einen Ausgabeauftrag erzeugt oder eine RSO-Parameterliste übergibt,
- den Transaktionscode eines Asynchron-Programms, wenn dieser FPUT-Aufruf einen Hintergrund-Auftrag erzeugt (ohne verteilte Verarbeitung),
- die Vorgangs-Id eines Auftragnehmer-Vorgangs, wenn dieser Hintergrund-Auftrag an einen Auftragnehmer-Vorgang gerichtet ist,
- den Namen der TAC-Queue, wenn die Nachricht an eine TAC-Queue gesendet werden soll,
- die Vorgangs-Id der fernen TAC-Queue, wenn diese Nachricht an eine ferne TAC-Queue gesendet werden soll.

KCMF / kcfn

im Feld KCMF/kcfn (bei Nachrichten an einen Asynchron-Vorgang bzw. eine TAC-Queue derselben Anwendung oder an einen LU6.1-Partner irrelevant):

- Leerzeichen im Zeilenmodus oder bei einem Auftrag an eine andere Anwendung ohne verteilte Verarbeitung oder beim Übergeben einer RSO-Parameterliste.

bei Nachrichten an OSI TP-Partner:

- Name der abstrakten Syntax der Nachricht. Leerzeichen stehen dabei für die abstrakte Syntax von UDT; in diesem Fall wird als Transfer Syntax BER verwendet, und die Encodierung der Nachricht übernimmt openUTM.Wird hier ein Wert ungleich Leerzeichen angegeben, dann muss die Nachricht an openUTM in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax, übergeben werden.

Nur auf BS2000-Systemen:

- ein Formatkennzeichen (im Formatmodus)

  Bei Nachrichten an RSO-Drucker:
  Wenn ein Format speziell für RSO-Drucker erstellt wurde, muss das Formatierungssystem FHS den Druckertyp nicht kennen, da FHS eine logische Nachricht erzeugt, die von RSO in die physikalische Nachricht umgewandelt wird.
  Andernfalls muss FHS den Druckertyp, wie er bei RSO generiert ist, unterstützen, da es sonst zum Formatierungsfehler kommt.
- Editprofil (bei Zeilenmodus oder einem RSO-Drucker)Ist die Nachricht an einen RSO-Drucker gerichtet, so wird nur der Parameter CCSNAME eines Editprofils ausgewertet. Der Zeichensatzname wird an RSO übergeben. Alle weiteren Parameter des Editprofils werden ignoriert, da es sich um VTSU-B Editoptionen handelt, die Nachricht aber von RSO aufbereitet wird.

KCDF

Im Feld KCDF (bei Hintergrund-Aufträgen und TAC-Queues irrelevant):
die Bildschirmfunktion, wenn der FPUT-Aufruf an einen LTERM-Partner gerichtet ist. Bei Aufträgen an andere Anwendungen ohne verteilte Verarbeitung oder bei Übergabe von RSO-Parameterlisten muss hier binär null angegeben werden.

Auf BS2000-Systemen musss binär null auch dann angegeben werden, wenn in KCMF/kcfn ein Editprofil oder ein #Format eingetragen ist.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Nachricht ein, die Sie ausgeben oder die RSO-Parameterliste, die Sie übergeben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

2. Parameter

als 2. Parameter: die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht oder RSO-Parameterliste lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLM die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode (siehe nächste Seite).

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim FPUT-Aufruf

Im Programm sind auswertbar:

000 Die Funktion wurde ausgeführt.

04Z Der Name in KCRN wechselt, ohne dass vorher FPUT NE gegeben wurde.

40Z openUTM kann die Funktion nicht durchführen (System- bzw. Generierungsfehler, Deadlock, langandauernde Sperren), siehe KCRCDC.

41Z In KCRN wurde der LTERM-Partner adressiert, der den laufenden Vorgang begonnen hat, oder FPUT wurde im ersten Teil des Anmelde-Vorgangs abgesetzt oder es wurde ein FPUT-Aufruf im Anmelde-Vorgang nach einem SIGN ON und vor dem PEND PS Aufruf abgesetzt.

42Z Der Eintrag in KCOM ist ungültig.

43Z Die Längenangabe in KCLM ist negativ bzw. ungültig.

44Z Der Wert in KCRN ist kein TAC eines Asynchron-Programms bzw. einer TAC-Queue, oder der TAC ist gesperrt bzw. verboten und ebenfalls kein Name eines LTERM-Partners bzw. der Name eines UPIC- oder HTTP-Clients, und keine gültige Vorgangs-Id (bei verteilter Verarbeitung), siehe KCRCDC.
Für die Dead Letter Queue (KDCDLETQ) dürfen keine Asynchron-Nachrichten erzeugt werden.

Nur auf BS2000-Systemen:
Bei KCOM = RP ist der Wert in KCRN kein RSO-Drucker oder die aktuelle RSO-Version unterstützt diese Funktion nicht.

45Z Der Eintrag in KCMF/kcfn ist unzulässig. Mögliche Ursachen:

- Das Formatkennzeichen in KCMF/kcfn ist nicht gültig.
- Ist die Nachricht an einen Partner gerichtet, mit dem über das OSI TP-Protokoll kommuniziert wird, dann bedeutet dieser Returncode, dass die im Feld KCMF/kcfn angegebene abstrakte Syntax für die Partner-Anwendung nicht generiert ist.

Nur auf BS2000-Systemen:

- Bei KCOM = RP: Kein Leerzeichen eingetragen
- Das Editprofil ist nicht generiert.
- Das Editprofil wechselt bei Nachrichtenteilen an Terminals.

47Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Eigenschaften des FPUT-/DPUT-Aufrufs

- FPUT- und DPUT-Aufrufe, die Teilprogramme an ein Alias-LTERM richten, werden wie folgt bearbeitet:
  - In einer LTERM-Gruppe ohne LTERM-Bündel werden FPUT-/DPUT-Aufrufe von openUTM über das PTERM gesendet, das dem Primary-LTERM zugeordnet ist.
  - In einer LTERM-Gruppe, deren Primary-LTERM das Master-LTERM eines LTERM-Bündels ist, weist openUTM beim Transaktionsende alle Asynchron-Nachrichten, die in dieser Transaktion an Alias-LTERMs der Gruppe gerichtet wurden, genau einem der Slave-LTERMs zu. Dieses Vorgehen garantiert, dass beim Empfänger die Reihenfolge der Nachrichten erhalten bleibt, die in einer Transaktion für eine LTERM-Gruppe erzeugt wurden.
- FPUT- und DPUT-Aufrufe, die Teilprogramme an das Master-LTERM richten, werden beim Transaktionsende einem der Slave-LTERMs zugewiesen.
- FPUT- und DPUT-Aufrufe können Teilprogramme auch direkt an das Primary-LTERM richten.
- Der Nachrichtenbereich wird beim Ausführen des Aufrufs durch openUTM nicht verändert.
- In einem Teilprogramm können mehrere Aufträge erzeugt werden; die zugehörigen Nachrichten können aus mehreren Teilen bestehen.
- Bei Ausgabe von +Formaten, *Formaten oder Nachrichten im Zeilenmodus können Sie auch Bildschirmausgabefunktionen verwenden, siehe Abschnitt „Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme)“.

  Nur auf BS2000-Systemen:
  - Wenn Sie #Formate verwenden, müssen Sie KCDF mit binär null versorgen, sonst reagiert openUTM mit 40Z.
  - Auch bei der Verwendung von Editprofilen reagiert openUTM mit 40Z, falls KCDF nicht mit binär null versorgt ist.
- Bei einem PEND ER/FR, PEND RS oder RSET werden die mit FPUT erzeugten Aufträge verworfen.
- In einem Dialog-Programm darf kein Ausgabeauftrag an den Client, mit dem das Programm gerade arbeitet, erzeugt werden (KCRN != KCLOGTER).
- Bei FPUT-Aufrufen an eine andere UTM-Anwendung, die als Transportsystem-Anwendung generiert ist, muss der TAC jeweils am Anfang des Nachrichtenbereichs stehen.
- Die Nachricht wird gegebenenfalls formatiert, bevor sie ausgegeben wird.
- Nachrichten werden aufbewahrt bis:
  - das angesprochene Teilprogramm oder die Druckerausgabe beendet ist oder, bei Aufträgen an ferne Asynchron-Vorgänge, die Übertragung erfolgreich abgeschlossen ist oder
  - die Nachricht mit KDCOUT am Terminal gelesen und eine neue Eingabe gemacht wurde (außer KDCLAST-Kommando) oder
  - die Nachricht aus einer TAC-Queue mit einem DGET-Aufruf gelesen und die den DGET beinhaltende Transaktion erfolgreich abgeschlossen wurde.
- Aufträge mit Nachrichten der Länge 0

  Erzeugt man eine Nachricht der Länge 0 (so genannte "leere Nachrichten"), so wird

- ein Hintergrund-Auftrag ausgeführt, d.h. der Asynchron-Vorgang wird gestartet, ohne dass er eine Nachricht erhält,
- bei einem Ausgabeauftrag im Formatmodus ein leeres Format ausgegeben,
- bei einer Nachricht für eine TAC-Queue eine leere Nachricht erzeugt, die mit einem DGET-Aufruf gelesen werden kann,
- ein Ausgabeauftrag an eine Transportsystem-Anwendung zwar angenommen, aber von openUTM zu einem späteren Zeitpunkt verworfen.

- Hintergrund-Aufträge an ein Asynchron-Programm der gleichen Anwendung: Jeder Hintergrund-Auftrag bewirkt, dass ein eigener Asynchron-Vorgang gestartet wird. Werden in einem Teilprogrammlauf mehrere vollständige Nachrichten an den gleichen TAC gesendet, wird für jede Nachricht ein eigener Asynchron-Vorgang gestartet
- Werden in einem Teilprogrammlauf mehrere vollständige Nachrichten an die gleiche TAC-Queue gesendet, muss jede Nachricht mit einem eigenen DGET FT-Aufruf gelesen werden.
- Wechselwirkungen zwischen FPUT- und DPUT-Aufrufen gibt es nicht, d.h. an ein bestimmtes Ziel können unabhängig voneinander DPUT-Aufrufe mit KCMOD = '"BLANK"' und FPUT-Aufrufe gesendet werden.
- Ausgabeaufträge, die für ein Terminal bestimmt sind, werden in die Message Queue eingehängt, und können vom Benutzer mit dem Kommando KDCOUT gelesen werden. Pro KDCOUT-Kommando wird genau eine Nachricht gelesen. Jede Nachricht kann nur einmal gelesen werden. Bei wiederholter Eingabe des KDCOUT-Kommandos wird die nächste Nachricht aus der Queue gelesen.

  Dass für ein Terminal Asynchron-Nachrichten vorliegen, wird dem Terminalbenutzer bei Transaktionsende durch eine Meldung in der Systemzeile mitgeteilt.

  Auf BS2000-Systemen kann diese Ankündigung unterdrückt werden, wenn bei der Konfiguration für den betreffenden LTERM-Partner ANNOAMSG=N angegeben wurde (Standardwert: ANNOAMSG=Y). Asynchron-Nachrichten werden dann sofort am Bildschirm angezeigt. Dadurch kann die Dialog-Führung gestört werden. Der Terminalbenutzer kann sich jedoch mit dem KDCDISP-Kommando den letzten Bildschirm wieder anzeigen lassen.
- Druckoptionen für RSO-Drucker (BS2000-Systeme)

  Wenn Sie Druckoptionen für Aufträge an RSO-Drucker verwenden, dann übergeben Sie zuerst mit FPUT RP die Liste mit den Druckoptionen, siehe Handbuch „RSO“. Danach geben Sie mit FPUT NT/NE den eigentlichen Druckauftrag.
- Behandeln von Teilnachrichten

- Teilnachrichten im Zeilenmodus werden zusammengefasst und als **eine** Nachricht an den LTERM-Partner ausgegeben. Die mit FPUT erzeugten Teilnachrichten werden von openUTM gesammelt und beim nächsten PEND-Aufruf abgeschlossen, falls der Teilprogrammlauf sie noch nicht mit FPUT NE abgeschlossen hat. Die Teilnachrichten werden bei Transaktionsende als eine Nachricht an den LTERM-Partner oder an die andere Anwendung gesendet.
- Bei formatierten Teilnachrichten an Terminals erzeugt jede Teilnachricht eine eigenständige Nachricht. Der Formatname in KCMF/kcfn darf dabei wechseln. An einem Terminal muss jedes Format (jeder Teilnachricht) mit einem KDCOUT-Kommando abgeholt werden. Jeder FPUT NT-Aufruf erzeugt eine eigene Nachricht. Deshalb ist es nicht möglich, mit FPUT NT-Aufrufen einen Bildschirm aus mehreren Teilformaten aufzubauen. Die Formate treffen in der Reihenfolge ein, in der sie erzeugt wurden.
- Bei Teilnachrichten an Drucker ist ein Wechsel zwischen formatierten Teilnachrichten und nicht-formatierten Teilnachrichten (im Zeilen-Modus) möglich. Bei Teilnachrichten an Terminals führt dieser Wechsel zum Abschluss der alten und zum Beginn einer neuen Nachricht.
- Bei FPUT NT-Aufrufen an lokale oder auch ferne Asynchron-Vorgänge bzw. lokale oder ferne TAC-Queues muss jede Teilnachricht mit einem eigenen FGET NT gelesen werden.
- Beim PEND wird die zuletzt mit FPUT aufgebaute Teilnachricht grundsätzlich als letzte Teilnachricht angenommen, auch wenn sie mit NT ausgegeben wurde.
- Die maximale Anzahl der FPUT NE-Aufrufe in einer Transaktion wird vom Generierungsparameter RECBUF der KDCDEF-Anweisung MAX begrenzt. Pro FPUT NE werden 30 Bytes in diesem Puffer belegt. Ist der Puffer voll, so wird der FPUT NE mit KCRCDC=K704 abgewiesen.
- Wechselt der Name in KCRN (d.h. der Empfänger der Nachricht) bei einer Folge von Teilnachrichten, ohne dass vorher ein FPUT NE gegeben wurde, wird eine Warnung (04Z) erzeugt und eine neue Nachricht begonnen. Die vorherige Nachricht (Teilnachrichtenfolge) wird abgeschlossen; d.h. wechselt mit einem folgenden FPUT der Name in KCRN noch einmal zurück zum ersten Empfänger, wird die erste Nachricht **nicht** fortgesetzt. Es wird eine neue Nachricht begonnen. Die Nachricht wird beim nächsten Sicherungspunkt an den Empfänger weitergeleitet. Ein paralleler Nachrichtenaufbau wie beim DPUT ist also nicht möglich.
- Nur auf BS2000-Systemen: Wechselt das Editprofil innerhalb einer Folge von Teilnachrichten, die an ein Terminal gerichtet sind, dann reagiert openUTM mit 45Z.

- Einfluss von Generierungsparametern auf den FPUT-Aufruf

  Die folgenden Hinweise beziehen sich auf die Generierung der UTM-Anwendung. Näheres zu den einzelnen Generierungsparametern finden Sie im openUTM-Handbuch „Anwendungen generieren“.

  - Bei Nachrichten an Terminals und Transportsystem-Anwendungen darf die Gesamtnachricht höchstens so groß sein wie der im Operanden NB der Steueranweisung MAX generierte Wert. Bei Nachrichten an andere Partner ist die Länge einer Teilnachricht auf 32767 beschränkt und die Länge der Gesamtnachricht unbegrenzt.
  - Beim Aktualisieren der KDCFILE mit dem UTM-Tool KDCUPD kann in der TRANSFER-Anweisung differenziert festgelegt werden, welche Nachrichten in die neue KDCFILE übernommen werden sollen (siehe auch openUTM-Handbuch „Anwendungen generieren“).
  - Nur auf BS2000-Systemen: Der Operand ANNOAMSG der KDCDEF-Steueranweisung LTERM legt für jeden LTERM-Partner fest, ob Asynchron-Nachrichten an dieses Terminal sofort ausgegeben oder mit einer Meldung angekündigt werden. Die Meldung erscheint in der Systemzeile.
- FPUT-Aufrufe bei verteilter Verarbeitung

- Im Auftraggeber-Vorgang ist der KDCS-Parameterbereich vor dem FPUT-Aufruf genauso zu versorgen wie bei Hintergrund-Aufträgen an ein Asynchron-Programm der eigenen Anwendung. Lediglich als Rufname im Feld KCRN ist die Vorgangs-Id anzugeben, die vorher beim APRO AM-Aufruf für den Auftragnehmer-Vorgang vergeben wurde.
- Nach Abschluss des Aufrufs FPUT NE (letzter Nachrichtenteil oder Gesamtnachricht) oder nach dem KDCS-Returncode 40Z wird die Vorgangs-Id im Auftraggeber-Vorgang freigegeben. Die gleiche Identifikation kann dann für eine neue Auftraggeber/Auftragnehmer-Beziehung dieses Vorgangs verwendet werden.
- Falls die Wartezeit für die Belegung einer Session bzw. Association bei der Generierung auf 0 gesetzt wurde und falls beim FPUT-Aufruf keine Verbindung zur Partner-Anwendung besteht, dann setzt openUTM nach dem FPUT-Aufruf die Returncodes 40Z in KCRCCC und KD13 in KCRCDC.

7.10 GTDA Lesen aus einem TLS

Mit dem Aufruf GTDA (get data) können Sie einen Block eines TLS (Terminal-spezifischer Langzeitspeicher) in den angegebenen Nachrichtenbereich einlesen. Der Blockname wird bei der Generierung vergeben (TLS-Anweisung bei KDCDEF).

Ein Teilprogramm eines Dialog-Vorgangs darf nur Blöcke des "eigenen" TLS lesen, also nur den TLS des LTERM-/LPAP-/OSI-LPAP-Partners, über den der Vorgang gestartet wurde.

Ein Teilprogrammlauf eines Asynchron-Vorgangs kann die Blöcke jedes LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners der UTM-Anwendung lesen.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**			
	KCOP	**KCLA**	**KCRN**	**KCLT**
Lesen aus einem TLS (im Dialog-Programm)	"GTDA"	Länge	Blockname	——
Lesen aus einem TLS (im Asynchron-Programm)	"GTDA"	Länge	Blockname	LTERM / LPAP / OSI-LPAP / Master-LPAP Name

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, in den openUTM die Nachricht einlesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"GTDA"
KCLA	Länge in Byte
KCRN	Blockname
KCLT	LTERM / LPAP / OSI-LPAP / Master-LPAP-Name / -

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_GDTA	(nb,la,rn,lt)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Blocklänge
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den GTDA-Aufruf ein:

KCOP

im Feld KCOP den Operationscode GTDA.

KCLA

im Feld KCLA die Länge, in der die Daten aus dem TLS übertragen werden sollen.

KCRN

im Feld KCRN den Namen des Blocks des TLS, aus dem openUTM übertragen soll.

KCLT

nur bei Asynchron-Programmen:
im Feld KCLT der Name des LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners, aus dessen TLS gelesen werden soll (von Dialog-Programmen wird dieses Feld nicht ausgewertet).

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht einlesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die gewünschten Daten.

KCRLM

im Feld KCRLM die tatsächliche Länge der Daten im TLS, damit das Programm Abweichungen von der Angabe in KCLA feststellen kann (wichtig, wenn KCLA kleiner angegeben wurde). Ausnahme: Bei KCLA = 0 wird in KCRLM immer 0 zurückgegeben.

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim GTDA-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout), siehe KCRCDC.
41Z	Der Aufruf wurde im ersten Teil des Anmelde-Vorgangs gegeben, obwohl dies in der Generierung nicht erlaubt wurde (SIGNON= in der MAX Anweisung).
43Z	Die Längenangabe in KCLA ist ungültig (z.B. negativ).
44Z	Der Name des Blocks in KCRN ist unbekannt oder ungültig.
46Z	Der LTERM/LPAP/OSI-LPAP/Master-LPAP-Name in KCLT ist ungültig (nur bei Asynchron-Programmen).
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Programm wurde kein INIT gegeben.

Eigenschaften des GTDA-Aufrufs

- Ein GTDA-Aufruf sperrt den Zugriff auf den angesprochenen TLS-Block für alle konkurrierenden Teilprogramme.
  Alle anderen TLS-Blöcke des angesprochenen LTERM-, LPAP- oder OSI-LPAP-Partners-Partners sind frei.
  Mit dem Aufruf UNLK kann der TLS-Block explizit entsperrt werden.
  Bei den Aufrufen PEND RE/FI/SP/FC/RS/ER/FR und RSET wird der TLS-Block entsperrt.
  Bei PEND PA/PR/KP und PGWT KP/PR bleibt die Sperre erhalten.

  Wie openUTM reagiert, wenn der gewünschte TLS-Block gesperrt ist, ist in Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“ beschrieben.

- Der TLS-Block wird in der tatsächlichen Länge übertragen, höchstens jedoch in der bei KCLA angegebenen Länge. War der Inhalt von KCLA beim GTDA-Aufruf > 0, so wird die tatsächliche Länge der Daten im TLS im Feld KCRLM zurückgegeben.

7.11 INFO Informationen abrufen

Mit den Varianten des Aufrufs INFO (information) können Sie die folgenden Informationen abrufen:

- INFO CD (**C**ar**d**), nur auf BS2000-Systemen
  Informationen, die auf dem Ausweis des Benutzers abgespeichert sind (nur wenn der Terminal-Benutzer beim Anmelden seine Berechtigung per Ausweisleser nachweisen muss) oder Kerberos-Informationen
- INFO DT (**D**ate/**T**ime)
  Datum und Uhrzeit des Anwendungsstarts und des Teilprogrammstarts
- INFO SI (**S**ystem **I**nformation)
  Systeminformationen (z.B. Name der Anwendung und des Rechners)
- INFO PC (**P**redecessor **C**onversation)
  Informationen über einen gekellerten Vorgang
- INFO LO (**LO**cale Information)
  Informationen über die Sprachumgebung des LTERM-Partners
- INFO CK (**C**hec**K)**
  den Returncode KCRCCC, der bei einem MPUT-, FPUT- oder PEND-Aufruf zu erwarten wäre

Diese Varianten des INFO-Aufrufs unterscheiden sich in der Bedeutung des 2. Parameters (Nachrichtenbereich), der beim INFO-Aufruf anzugeben ist.

Für die Strukturierung des Nachrichtenbereichs stellt openUTM Sprach-spezifische Datenstrukturen zur Verfügung: für COBOL im COPY-Element KCINFC, für C/C++ in der Include-Datei *kcinf.h.*

Versorgen des KDCS-Parameterbereichs (1. Parameter) und des 2. Parameters

Die folgende Tabelle zeigt die fünf Arten des INFO-Aufrufs und die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich				Bedeutung des 2. Parameters
	KCOP	KCOM	KCLA	KCLT	
BS2000-Systeme: Lesen der Ausweis- oder - Kerberos-Informationen	"INFO"	"CD"	Länge	—	Bereich für Ausweis- oder Kerberos-Informationen
Ermitteln von Datum und Uhrzeit von Anwendungs- und Programmstart	"INFO"	"DT"	30	—	Bereich für Datum und Zeit, siehe Datenstruktur für INFO DT.
Informationen über Sprachumgebung des LTERM-Partners	"INFO"	"LO"	68	LTERM-Name	Bereich für angeforderte Namen, siehe Datenstruktur für INFO LO.
Informationen über einen gekellerten Vorgang	"INFO"	"PC"	39	—	Bereich für angeforderte Informationen, siehe Datenstruktur für INFO PC.
Ermitteln von Systeminformationen	"INFO"	"SI"	180	—	Bereich für angeforderte Namen, siehe Datenstruktur für INFO SI.
Überprüfen eines UTM-Aufrufs	"INFO"	"CK"	—	—	Parameterbereich, wie er beim zu prüfenden Aufruf versorgt werden müsste.

Wegen der verschiedenen Bedeutungen des 2. Parameters wird der INFO-Aufruf hier in zwei Formaten dargestellt:

Format 1: INFO CD/DT/LO/PC/SI (siehe nächste Seiten).

Format 2: INFO CK (siehe "INFO CK-Aufruf").

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"INFO"
KCOM	"CD"/"DT"/"LO"/"PC"/"SI"
KCLA	Länge des Nachrichtenbereichs in Bytes
KCLT	Name des LTERM-Partners / —

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_INFOCD/KDCS_INFODT	(nb,kcla)
KDCS_INFOLO	(nb,kcla,kclt)
KDCS_INFOPC/KDCS_INFOSI	(nb,kcla)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Blocklänge
KCRCCC	Returncode
KCRCDC	interner Returncode

Zum Lesen der Ausweisinformation, zum Ermitteln von Datum, Zeit und Systeminformationen, zum Informieren über den Vorgänger oder über die Sprachumgebung des LTERM-Partners tragen Sie im KDCS-Parameterbereich für den INFO-Aufruf ein:

KCOP

im Feld KCOP den Operationscode INFO

KCOM

im Feld KCOM die gewünschte Funktion des Aufrufs:

- CD zum Lesen der Ausweis- oder Kerberos-Information (CARD), nur auf BS2000-Systemen
- DT zum Ermitteln von Uhrzeit und Datum des Starts der Anwendung und des Teilprogramms (DATE/TIME),
- LO zum Informieren über die Sprachumgebung des LTERM-Partners.
- PC zum Informieren über den Vorgänger im Stapel,
- SI zum Ermitteln von Systeminformation,

KCLA

im Feld KCLA die Länge des Nachrichtenbereichs, in den openUTM die Information hinterlegen soll. Die Länge ist als Anzahl Bytes anzugeben. openUTM überträgt die Information maximal in der Länge KCLA.

nur bei INFO LO:

KCLT

im Feld KCLT der Name des LTERM-Partners, dessen Sprachumgebung ermittelt werden soll.

Wird KCLT vor dem Aufruf mit binär null belegt, dann übergibt openUTM die Daten des LTERM-Partners, über den der Vorgang gestartet wurde.

Nur auf BS2000-Systemen:
Wird in KCLT der LTERM-Partner angegeben, der zum Teilprogrammlauf gehört, dann übergibt openUTM zusätzlich Informationen über das zugehörige physische Terminal (PTERM).

Beim KDCS-Aufruf geben Sie an:

1. Parameter

Die Adresse des KDCS-Parameterbereichs.

2. Parameter

Die Adresse des Nachrichtenbereichs, in den openUTM die Informationen schreiben soll. openUTM übergibt die angeforderten Informationen in einer festgelegten Struktur und stellt dafür Sprach-spezifische Datenstrukturen zur Verfügung: für COBOL im COPY-Element KCINFC, für C/C++ in der Include-Datei *kcinf.h.*

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die Information in der angegebenen Länge.

KCRLM

im Feld KCRLM die tatsächliche Länge der übertragenen Information.
Bei KCRCCC >= 40Z ist die Länge 0.

Nur auf BS2000-Systemen:
Bei INFO CD gibt KCRLM die Länge der Kerberos-Information an, die in den Empfangsbereich übertragen wurde.

KCRCCC

im Feld KCRCCC den KDCS-Returncode. 0

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim INFO CD/DT/LO/PC/SI-Aufruf

Im Programm sind auswertbar:

000	Die angeforderte Information wurde in der vollen Länge in den Nachrichtenbereich übertragen.
01Z	Es wurden Informationen übertragen, der Nachrichtenbereich ist jedoch zu kurz, die Information wurde abgeschnitten.
09Z	*nur auf BS2000-Systemen:* bei INFO CD: Der Kerberos-Dialog hat einen Fehler geliefert oder die Kerberos-Information wird verkürzt zurückgegeben, weil sie größer war als der bei MAX PRINCIPAL-LTH generierte Wert. KCRLM zeigt an, in welcher Länge die Kerberos-Information in den Empfangsbereich übertragen wurde.
40Z	Generierungs- oder Systemfehler, siehe KCRCDC.
41Z	Aufruf INFO mit KCOM = CD/PC in Asynchron-Programm nicht erlaubt.
42Z	KCOM ungültig.
43Z	KCLA ungültig.
46Z	nur bei INFO LO: der im Feld KCLT angegebene LTERM-Name ist ungültig.
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.
49Z	nur bei INFO LO: beim Aufruf waren nicht verwendete Parameter nicht mit binär null versorgt.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Teilprogramm wurde noch kein INIT gegeben.

Eigenschaften des INFO CD-Aufrufs (BS2000-Systeme)

Mit dieser Funktion liest der INFO-Aufruf die Ausweisinformation, die beim Anmelden gespeichert wurde, in den angegebenen Nachrichtenbereich, wenn der Benutzer bei der Anmeldung seine Berechtigung per Magnetstreifenkarte nachgewiesen hat. Näheres siehe auch Abschnitt „Unterstützung von Ausweislesern auf BS2000-Systemen“.

Die gespeicherte Kerberos-Information (= Name des Kerberos Principal) kann gelesen werden, wenn ein Benutzer angemeldet ist, der seine Berechtigung per Kerberos nachgewiesen hat oder für den Client ein Kerberos-Dialog durchgeführt wurde und sich nach dem Kerberos-Dialog kein Benutzer mehr mit Magnetstreifenkarte angemeldet hat.

Der INFO CD-Aufruf zum Lesen der Ausweis- oder Kerberos-Information ist nur in Dialog-Programmen zulässig.

Um die Ausweisinformation im Programm abrufen zu können, müssen bei der Generierung folgende Operanden versorgt werden:

- für die aktuelle Benutzerkennung in der KDCDEF-Steueranweisung USER der Operand CARD= und
- der Operand CARDLTH= in der KDCDEF-Steueranweisung MAX.

ℹ Im Feld KCAUSWEIS/kccard im KB-Kopf setzt openUTM das Kennzeichen "A", wenn bei der letzten Eingabe der Ausweis gesteckt hat.

Eigenschaften des INFO DT-Aufrufs

Mit dieser Funktion schreibt der INFO-Aufruf den Startzeitpunkt von Anwendung und Teilprogramm in der Länge von 30 Byte in den angegebenen Nachrichtenbereich. Die einzelnen Bytes haben folgende Bedeutung:

Feldname COBOL	Feldname C/C++	Byte	Bedeutung der Information
KCDATAS	—	1 - 6	Datum des Anwendungsstarts in der Form *ddmmyy*. Dabei bedeuten:
KCTAGAS	as_day	1 - 2	*dd* - Tag (Wertebereich 01 - 31)
KCMONAS	as_mon	3 - 4	*mm* - Monat (Wertebereich 01 - 12)
KCJHRAS	as_year	5 - 6	*yy* - Jahr (Wertebereich 00 - 99)
KCTJHAS	as_doy	7 - 9	Tag des Anwendungsstarts (Industrietag, Wertebereich 001 - 366)
KCUHRAS	—	10 - 15	Uhrzeit des Anwendungsstarts in der Form *hhmmss*. Dabei bedeuten:
KCSTDAS	as_hour	10 -11	*hh* - Stunde (Wertebereich 00 - 23)
KCMINAS	as_min	12 - 13	*mm* - Minute (Wertebereich 00 - 59)
KCSEKAS	as_sec	14 - 15	*ss* - Sekunde (Wertebereich 00 - 59)
KCDATAK	—	16 - 21	Datum des Teilprogrammstarts in der Form *ddmmyy*. Dabei bedeuten:
KCTAGAK	ps_day	16 - 17	*dd* - Tag (Wertebereich 01 - 31)
KCMONAK	ps_mon	18 - 19	*mm* - Monat (Wertebereich 01 - 12)
KCJHRAK	ps_year	20 - 21	*yy* - Jahr (Wertebereich 00 - 99)
KCTJHAK	ps_doy	22 - 24	Tag des Teilprogrammstarts (Industrietag, Wertebereich 001 - 366)
KCUHRAK	—	25 - 30	Uhrzeit des Teilprogrammstarts in der Form *hhmmss*. Es bedeuten:
KCSTDAK	ps_hour	25 - 26	*hh* - Stunde (Wertebereich 00 - 23)
KCMINAK	ps_min	27 - 28	*mm* - Minute (Wertebereich 00 - 59)
KCSEKAK	ps_sec	29 - 30	*ss* - Sekunde (Wertebereich 00 - 59)

Eigenschaften des INFO LO-Aufrufs (BS2000-Systeme)

Mit der Variante INFO LO des INFO-Aufrufs kann der Teilprogrammlauf Information über das Locale von LTERM-Partnern und Anwendung anfordern. Das Locale eines LTERM-Partners legt die Sprachumgebung der Clients fest. Das Locale der Anwendung legt die Standardeinstellung der Sprachumgebung für die Anwendung fest.

Der Aufruf liefert folgende Daten zurück:

- das Locale des LTERM-Partners, dessen Name im Feld KCLT angegeben ist
- das Locale der Anwendung

Wird KCLT vor dem Aufruf mit binär null belegt, dann übergibt openUTM die Daten des LTERM-Partners, über den der Vorgang gestartet wurde.

Wird der Aufruf in einem Dialog-Vorgang ausgeführt und wird im Feld KCLT der Name des LTERM-Partners angegeben, der zum Teilprogrammlauf gehört, dann übergibt openUTM zusätzlich folgende Daten über das zugehörige physische Terminal (PTERM):

- die Anzahl der erweiterten ISO-Zeichensätze, die das Terminal bzw. der Drucker unterstützt,
- die ISO-Variantennummern aller Zeichensätze, die unterstützt werden,
- den Standard-Anwenderzeichensatz, der der BS2000-Benutzerkennung zugeordnet ist, auf der die UTM-Anwendung läuft.

Beim INFO LO-Aufruf stellt openUTM die Daten im Nachrichtenbereich gemäß folgender Struktur zur Verfügung:

Feldname	Byte	Bedeutung des Inhalts
KCLTLANG	1 - 2	Sprachkennzeichen des LTERM-Partners
KCLTTERR	3 - 4	Territorialkennzeichen des LTERM-Partners
KCLTCCSN	5 - 12	CCS-Name des LTERM-Partners
	13 - 20	Leerzeichen
KCAPLANG	21 - 22	Sprachkennzeichen der Anwendung
KCAPTERR	23 - 24	Territorialkennzeichen der Anwendung
KCAPCCSN	25 - 32	CCS-Name der Anwendung
	33 - 40	Leerzeichen
KCDEFCCS	41 - 48	Anwender-Standardzeichensatz der BS2000-Benutzerkennung, auf der die UTM-Anwendung läuft

Feldname	Byte	Bedeutung des Inhalts
Information über das angeschlossene Terminal. (Wird nur in Dialog-Vorgängen ausgegeben, wenn KCLT den Namen des zum Vorgang gehörenden LTERM-Partners enthält. Falls es sich nicht um ein Terminal handelt, sind die Werte mit X'00' belegt.)		
KCCCSNO	49	Anzahl der vom Terminal unterstützten erweiterten ISO-Zeichensätze
KCHSET1	50	Variantennummer des 1. unterstützten ISO-Zeichensatzes
KCHSET2, KCHSET3 ... bis KCHSET16	51 - 65	Variantennummern weiterer ISO-Zeichensätze, die das physische Terminal unterstützt. Länge der einzelnen Felder ist 1 Byte.
	65 - 68	Leerzeichen

Eigenschaften des INFO LO-Aufrufs (Unix-, Linux- und Windows-Systeme)

Mit der Variante INFO LO des INFO-Aufrufs kann der Teilprogrammlauf Information über die Sprache anfordern, die für den LTERM-Partner aktuell eingestellt ist.

Mit Hilfe dieser Funktion können Sie die Mehrsprachigkeit Ihrer Teilprogramme realisieren. Im Teilprogrammlauf kann die für einen LTERM-Partner eingestellte Sprache mit INFO LO abgefragt werden und dann die Nachricht in dieser Sprache gesendet werden.

INFO LO liefert folgende Information zurück:

- Sprach- und Territorialkennzeichen sowie die $LANG-Variable des in KCLT angegebenen LTERM-Partners. Ausgegeben wird die aktuell eingestellte $LANG-Variable der Benutzerkennung, unter der der zugehörige Dialog-Terminalprozess gestartet wurde.
- Sprach- und Territorialkennzeichen sowie die $LANG-Variable der UTM-Anwendung. Ausgegeben wird die $LANG-Variable der Benutzerkennung, unter der die Anwendung gestartet wurde.

Sprach- und Territorialkennzeichen werden zur Laufzeit des Programms aus der $LANG-Variablen gewonnen. Beispiel: Aus $LANG=En_US.ASCII erzeugt openUTM das Sprachkennzeichen En und das Territorialkennzeichen US.

Der Aufruf INFO LO liefert Daten gemäß folgender Struktur:

Feldname	Byte	Bedeutung des Inhalts
KCLTLANG	1 - 2	Sprachkennzeichen des LTERM-Partners
KCLTTERR	3 - 4	Territorialkennzeichen des LTERM-Partners
KCLTNLSL	5 - 20	$LANG-Variable des angegebenen LTERM-Partners
KCAPLANG	21 - 22	Sprachkennzeichen der Anwendung
KCAPTERR	23 - 24	Territorialkennzeichen der Anwendung
KCAPNLSL	25 - 40	$LANG-Variable der Anwendung
	41 - 68	Leerzeichen

Eigenschaften des INFO PC-Aufrufs

Mit diesem Aufruf können Sie Informationen, über einen gekellerten Vorgang abfragen. Wenn der aktuelle Vorgang durch Vorgangs-Kellerung gestartet wurde, liefert der INFO PC-Aufruf über den Vorgänger-Vorgang (den Vorgang der im Stapel unmittelbar unter dem aktuellen Vorgang liegt) z.B. die folgenden Informationen:

- Datum und Uhrzeit der letzten Verarbeitung
- das Formatkennzeichen der letzten Bildschirmausgabe
- den nächsten TAC
- den Vorgangs-TAC

openUTM schreibt diese Informationen in der Länge von insgesamt 39 Byte wie folgt in den angegebenen Bereich.

Feldname	Byte	Bedeutung der Information
KCPFN	1 - 8	Formatkennzeichen der letzten Bildschirmausgabe
KCPNXTAC	9 - 16	Name des nächsten TACs
KCPCVTAC	17 - 24	Name des Vorgangs-TACs
KCPLDATE[1]	25 - 33	Datum der letzten Verarbeitung in der Form ddmmyy. Es bedeuten:
KCPLDAY	25 - 26	dd - Tag (Wertebereich 01 - 31)
KCPLMON	27 - 28	mm - Monat (Wertebereich 01 - 12)
KCPLYEAR	29 - 30	yy - Jahr (Wertebereich 00 - 99)
KCPLDOY	31 - 33	Industrietag der letzten Verarbeitung, Wertebereich 001 - 366
KCPLTIME[1]	34 - 39	Uhrzeit der letzten Verarbeitung in der Form hhmmss. Es bedeuten:
KCPLHOUR	34 - 35	hh - Stunde (Wertebereich 00 - 23)
KCPLMIN	36 - 37	mm - Minute (Wertebereich 00 - 59)
KCPLSEC	38 - 39	ss - Sekunde (Wertebereich 00 - 59)

[1]Für C/C++ sind die zusammenfassenden Felder KCPLDATE und KCPLTIME nicht definiert, wohl aber die spezifischen Felder für Tag/Monat/Jahr/Industrietag/Stunde/Minute/Sekunde.

Liegt keine Kellerung vor, so gibt openUTM Leerzeichen zurück.

Eigenschaften des INFO SI-Aufrufs

Mit diesem Aufruf können Sie Informationen zu Anwendung und System abfragen, z.B. den Namen der Anwendung und des Rechners, auf dem die Anwendung läuft.
In Dialog-Vorgängen liefert INFO SI zusätzlich den PTERM-Namen, den Rechnernamen und den Anwendungsnamen des Kommunikationspartners.

openUTM schreibt diese Informationen wie folgt in den angegebenen Bereich. Byte 17 - 40 werden in Asynchron-Vorgängen mit Leerzeichen versorgt.

Feldname	Byte	Bedeutung der Information
KCAPPLNM /applnam	1 - 8	Name der UTM-Anwendung.
KCHOSTNM /hostname	9 - 16	*BS2000-Systeme:* • Name des Rechners, auf dem die Anwendung abläuft.
		Unix-, Linux- und Windows-Systeme: • Stand-alone-Anwendungen: Name des mit MAX HOSTNAME generierten Hosts • UTM-Cluster-Anwendungen: Name des mit CLUSTER-NODE VIRTUAL-HOST generierten Hosts
KCPTRMNM	17 - 24	im Dialog-Vorgang: • bei verteilter Verarbeitung über LU6.1: CON-Name des Kommunikationspartners, • bei verteilter Verarbeitung über OSI TP: OSI-CON-Name des Kommunikationspartners, • sonst PTERM-Name des Kommunikationspartners. im Asynchron-Vorgang: Leerzeichen.
KCPRONM	25 - 32	im Dialog-Vorgang: • OSI TP-Auftragnehmer-Vorgang: Leerzeichen, • sonst: Rechnername des Kommunikationspartners. im Asynchron-Vorgang: Leerzeichen.
KCBCAPNM	33 - 40	im Dialog-Vorgang: • OSI TP Auftragnehmer-Vorgang: ACCESS-POINT-Name, über den die Association zum Partner aufgebaut ist. • sonst: BCAMAPPL-Name, über den die Verbindung zum Partner aufgebaut ist. im Asynchron-Vorgang: Leerzeichen.
KCVERS	41 - 46	openUTM-Version V*nn.nx* (z.B. V07.0A).

Feldname	Byte	Bedeutung der Information
KCIVER	47 - 48	Versionsnummer der KDCS-Schnittstelle [1]
KCIVAR	49	"B" für BS2000-Systeme, "X" für Unix- und Linux-Systeme oder "N" für Windows-Systeme.
KCFILL1	50	zur Zeit nicht belegt.
KCLANG	51 - 52	*Nur auf Unix-, Linux- und Windows-Systemen:* • Sprachkennzeichen, z.B: "EN" für Englisch (Der Wert des Sprachkennzeichens entspricht den ersten beiden Bytes der $LANG-ariablen).
KCHSTNML	53 - 116	*BS2000-Systeme:* • langer Name des Rechners, auf dem die Anwendung abläuft.
		Unix-, Linux- und Windows-Systeme: • Stand-alone-Anwendungen: langer Name des mit MAX HOSTNAME generierten Hosts • UTM-Cluster-Anwendungen: langer Name des mit CLUSTER-NODE VIRTUAL-HOST generierten Hosts.
KCPRONML	117 - 180	im Dialog-Vorgang: • OSI TP Auftragnehmer-Vorgang: Leerzeichen, • sonst: langer Rechnername des Kommunikationspartners. im Asynchron-Vorgang: Leerzeichen.

[1]Die Versionsnummer zeigt den Stand der verfügbaren Funktionserweiterungen an, unabhängig von der Produktvariante. Damit wird die Erstellung von UTM-Anwendungsprogrammen unterstützt, die in verschiedenen Produktvarianten und openUTM-Versionen ablaufen. Bei funktionaler Erweiterung der KDCS-Schnittstelle wird die Versionsnummer erhöht. Für openUTM V7.0 hat sie den Wert 9.
Die Versionsnummer der KDCS-Schnittstelle wird nicht erhöht, wenn die funktionale Erweiterung der KDCS-Schnittstelle Datenstrukturen von KDCS-Aufrufen betrifft, die eine eigene Versionsnummer haben.

7.11.1 INFO CK-Aufruf

Mit dieser Funktion überprüft der INFO-Aufruf für einen MPUT-, FPUT- oder PEND-Aufruf, ob die Angabe in KCRN für den jeweiligen Aufruf zulässig ist. openUTM liefert im Feld KCRINFCC den KDCS-Returncode, der bei dem geprüften Aufruf zu erwarten ist. Im einzelnen überprüft INFO CK folgende Aufrufe:

- MPUT NT/NE
- FPUT NT/NE
- PEND PA/PR/KP/SP/RE/FC

Mit dieser Funktion können Sie z.B. vor einem PEND RE-Aufruf prüfen lassen, ob der beabsichtigte Folge-TAC für diesen Aufruf zulässig ist, ohne einen Vorgangs-Abbruch zu riskieren.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"INFO"
KCOM	"CK"
Nachrichtenbereich	
	KDCS-Parameterbereich des zu prüfenden Aufrufs

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_INFOCK	(nb)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRINFCC	Returncode KCRCCC des geprüften Aufrufs
KCRCCC	Returncode
KCRCDC	interner Returncode

Zum Überprüfen des KDCS-Parameterbereiches für einen UTM-Aufruf tragen Sie für den INFO-Aufruf im KDCS-Parameterbereich ein:

KCOP

im Feld KCOP den Operationscode INFO.

KCOM

im Feld KCOM die gewünschte Funktion des Aufrufs: CK zum Überprüfen des KDCS-Parameterbereiches für einen MPUT-, FPUT-, oder PEND-Aufruf.

Nachrichtenbereich

den zu prüfenden KDCS-Parameterbereich entsprechend den für den jeweiligen Aufruf (MPUT, FPUT, PEND) geltenden Vorschriften.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

Die Adresse des KDCS-Parameterbereichs.

2. Parameter

Die Adresse des Bereichs, in dem der KDCS-Parameterbereich des zu prüfenden Aufrufs steht.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRINFCC

im Feld KCRINFCC den Returncode, der bei dem geprüften Aufruf im Feld KCRCCC zu erwarten ist. Diesen Returncode trägt openUTM nur ein, wenn der INFO-Aufruf ordnungsgemäß verlaufen ist (der zum INFO-Aufruf gehörende Returncode ist = 000).KCRINFCC hat den Wert 78Z, wenn der zu prüfende Aufruf in der angegebenen Funktionsausprägung (Angabe in KCOP und KCOM im 2. Parameterbereich) nicht vom INFO-Aufruf unterstützt wird.

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode des geprüften Aufrufs (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim INFO CK-Aufruf

Im Programm sind auswertbar:

000	Der im zweiten Parameter angegebene Aufruf wurde geprüft.
40Z	Generierungs- oder Systemfehler, siehe KCRCDC.
42Z	KCOM des INFO-Aufrufs ungültig.

47Z Adresse des 2. Parameters fehlt oder ist ungültig.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Teilprogramm wurde noch kein INIT gegeben.

Eigenschaften des INFO-CK-Aufrufs

Falls KCCARD angegeben wurde, überprüft der INFO CK-Aufruf bei MPUT, ob das Terminal, an das die Nachricht gerichtet ist, mit Ausweisleser generiert ist, bzw. ob der USER bereits mit Ausweis angemeldet ist.

Auf BS2000-Systemen wird eine solche Prüfung auch durchgeführt, wenn ein entsprechendes Editprofil angegeben wurde.

7.12 INIT Initialisieren eines Teilprogramms

Mit dem Aufruf INIT (initialize program) melden Sie ein Teilprogramm bei openUTM an. Dabei gibt es folgende Varianten:

- INIT (ohne Angabe in KCOM)
  Teilprogramm initialisieren
- INIT PU (**P**rogram **U**nit)
  Teilprogramm initialisieren und zusätzliche Informationen anfordern.
- INIT MD (**M**o**d**ifiy)
  Teilprogramm initialisieren und Größe des KB-Programmbereichs ändern.

Für diese Varianten gilt Folgendes:

- Der INIT-Aufruf leitet die Zusammenarbeit des Teilprogrammlaufs mit openUTM ein. Er ist der erste KDCS-Aufruf, der in einem Teilprogrammlauf erlaubt ist, d.h. vor dem INIT-Aufruf dürfen Sie keine anderen KDCS- oder Datenbank-Aufrufe geben.
- Der Aufruf INIT bzw. INIT PU darf je Teilprogrammlauf nur einmal abgesetzt werden
- INIT MD darf in einem Teilprogramm mehrmals angegeben werden. Wenn Sie den Aufruf INIT MD als ersten INIT-Aufruf im Teilprogramm angeben, wird er behandelt wie ein INIT ohne Operationsmodifikation.
- Der Kommunikationsbereich (KB) und der Standard Primäre Arbeitsbereich (SPAB) dürfen zwischen dem Start des Teilprogrammlaufs und dem ersten INIT-Aufruf nicht verwendet werden.
- Nach dem ersten INIT stellt openUTM dem Teilprogrammlauf den gesamten Kommunikationsbereich (KB) einschließlich des KB-Programmbereichs in der in KCLKBPRG/ kclcapa angegebenen Länge zur Verfügung.
- Der Aufruf INIT MD dient dazu, die Länge des KB-Programmbereichs im Teilprogrammlauf nachträglich zu verändern. Dies ist z.B. dann sinnvoll, wenn sich erst im Lauf der Verarbeitung ergibt, wie groß der KB-Programmbereich ist, den openUTM beim PEND-Aufruf sichern soll.
  Beispiel: Im KB-Programmbereich sollen Daten gesichert werden, die aus einer Datenbank gelesen werden. Wird die Länge des KB-Programmbereichs durch INIT MD an die gelesene Datenmenge angepasst, dann hat dies den Vorteil, dass openUTM am Sicherungspunkt nicht unnötig viele Daten oder zuwenig Daten sichert.
- Falls Sie den Aufruf INIT mit der Operationsmodifikation PU (**P**rogram **U**nit) verwenden, stellt openUTM dem Teilprogramm zusätzlich im Nachrichtenbereich Informationen über Anwendung, System und Kommunikationspartner zur Verfügung.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich				
	KCOP	KCOM	KCLKBPRG /kclcapa	KCLPAB /kclspa	KCLI
Initialisieren des Teilprogramms	"INIT"	—	Länge des KB-Programmbereichs (wenn verwendet)	Länge des SPAB (wenn verwendet)	—
Initialisieren des Teilprogramms und Anfordern von Informationen	"INIT"	"PU"	Länge des KB-Programmbereichs (wenn verwendet)	Länge des SPAB (wenn verwendet)	Länge des Nachrichtenbereichs
Länge des KB-Programmbereichs ändern	"INIT"	"MD"	Länge des KB-Programmbereichs	—	—

Die Länge des KB-Programmbereichs darf nicht größer sein als die Maximallänge, die für diese Anwendung bei der Generierung festgelegt wurde (Operand KB in der MAX-Anweisung bei KDCDEF).

Die Länge des SPAB darf nicht länger sein als bei der Generierung vorgesehen (Operand SPAB in der MAX-Anweisung bei KDCDEF).

Versorgung des 2. Parameters (nur notwendig bei INIT PU)

Hier geben Sie die Adresse des Nachrichtenbereichs an, in den openUTM die angeforderte Information schreiben soll.

Für die Strukturierung des Nachrichtenbereichs stehen Ihnen Sprach-spezifische Datenstrukturen zur Verfügung. Für COBOL sind diese im COPY-Element KCINIC definiert, für C/C++ in der Include-Datei *kcini.h.*

Im Header der Datenstruktur geben Sie die Versionsnummer der Struktur an und wählen aus, welche Information openUTM zurückliefern soll. Die Beschreibung der übrigen Felder (Rückgabeinformationen) finden Sie im Abschnitt „Struktur des Nachrichtenbereichs bei INIT PU".

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"INIT"
KCOM	— /"PU"/"MD"
KCLKBPRG/kclcapa	Länge in Byte
KCLPAB/kclspa	— /Länge in Byte
KCLI	— / Länge in Byte (bei INIT PU)

Versorgen des Headers des Nachrichtenbereichs (nur notwendig bei INIT PU)	
Feldname im Nachrichtenbereich	Inhalt
KCVER/if_ver	Versionsnummer (7)
KCDATE/dattim_info	Anfordern von Datum und Uhrzeit (Y/N)
KCAPPL/appl_info	Anfordern von Anwendungsinformation (Y/N)
KCLOCALE/locale_info	Anfordern von Locale Information (Y/N)
KCOSITP/ositp_info	Anfordern von OSI TP Informationen (Y/N)
KCENCR/encr_info	Anfordern von Verschlüsselungs-Informationen (Y/N)
KCMISC/misc_info	Anfordern von verschiedenen Informationen (Y/N)
KCHTTP/http_info	Anfordern von HTTP Informationen (Y/N)

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	— / Nachrichtenbereich (bei INIT PU)

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_INIT	(kclcapa,kclspa)
KDCS_INITPU	(nb,kclcapa,kclspa,kcli)
KDCS_INITMD	(kclcapa)

Rückgaben von openUTM	
Nachrichtenbereich (nur bei INIT PU)	**Inhalt**
	INIT PU Informationen
KB-Kopfbereich	
	aktuelle Vorgangs-Daten
Feldname im KB-Rückgabebereich	
KCRLM (nur bei INIT PU)	Länge der übergebenen Informationen
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Formatkennzeichen/Leerzeichen
KCRPI	Vorgangs-Id/Reset-Id/Leerzeichen
KB-Programmbereich	
KCKBPRG/kclcapa	Daten

Im KDCS-Parameterbereich machen Sie für den INIT-Aufruf folgende Einträge:

KCOP

im Feld KCOP den Operationscode INIT.

KCOM

im Feld KCOM die Operationsmodifikation:

- PU: wenn openUTM im Nachrichtenbereich zusätzliche Informationen bereitstellen soll
- MD: wenn die Länge des KB-Programmbereichs verändert werden soll

KCLKBPRG/kclcapa

im Feld KCLKBPRG/kclcapa die Länge des KB-Programmbereichs in Bytes (wenn verwendet). Sie darf nicht größer sein, als bei der Generierung vorgesehen (Operand KB in der MAX-Anweisung), sonst wird der generierte Wert genommen.

nur bei INIT oder INIT PU:

KCLPAB/kclspa

im Feld KCLPAB/kclspa die Länge des im Teilprogrammlauf verwendeten Standard Primären Arbeitsbereichs (SPAB) in Bytes. Sie darf nicht größer sein, als bei der Generierung vorgesehen (Operand SPAB in der MAX-Anweisung).

nur bei INIT PU:

KCLI

im Feld KCLI die Länge des Nachrichtenbereichs, in den openUTM die Information übertragen soll. Die Länge geben Sie als Anzahl Bytes an. Die von openUTM in den Nachrichtenbereich übertragene Information hat maximal die Länge KCLI. Um die gesamte verfügbare Information zu erhalten, müssen Sie in der aktuellen Schnittstellenversion den Wert 372 angeben.

Versorgen des Headers des Nachrichtenbereichs (nur notwendig bei INIT PU):

KCVER/if_ver

Im Feld KCVER/if_ver geben Sie die Versionsnummer der Datenstruktur an, die aktuelle Version ist 7.

KCDATE/dattim_info

Das Feld KCDATE/dattim_info versorgen Sie mit Y, falls Sie Informationen über Datum und Uhrzeit des Starts der Anwendung und des Teilprogrammlaufs haben wollen, sonst geben Sie N an.

KCAPPL/appl_info

Das Feld KCAPPL/appl_info versorgen Sie mit Y, wenn Sie Informationen über Anwendung, System und Kommunikationspartner anfordern, sonst geben Sie N an.

KCLOCALE/locale_info

Das Feld KCLOCALE/locale_info versorgen Sie mit Y, wenn Sie Informationen über die Sprachumgebung der Benutzerkennung anfordern, sonst geben Sie N an.

KCOSITP/ositp_info

Das Feld KCOSITP/ositp_info versorgen Sie mit Y, wenn Sie OSI TP-spezifische Informationen benötigen, sonst geben Sie N an.

KCENCR/encr_info

Das Feld KCENCR/encr_info versorgen Sie mit Y, wenn Sie Informationen über Verschlüsselungsverfahren benötigen, die zwischen dem Client und der UTM-Anwendung gefahren werden, sonst geben Sie N an. (Der Verschlüsselungsmechanismus kann vereinbart werden. Siehe openUTM-Handbuch „Anwendungen generieren“.)

KCMISC/misc_info

Das Feld KCMISC/misc_info versorgen Sie mit Y, wenn Sie verschiedene Informationen (z.B. Anzahl der Asynchron-Nachrichten in der Queue des Benutzers, Passwortgültigkeit, Zeitpunkt der letzten Anmeldung) benötigen, sonst geben Sie N an.

KCHTTP/http_info

Das Feld **KCHTTP/http_info** versorgen Sie mit Y, wenn Sie HTTP-spezifische Informationen benötigen, sonst geben Sie N an.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   (nur notwendig bei INIT PU): Die Adresse des Nachrichtenbereichs, in den openUTM die Information schreiben soll (siehe "Versorgung des 2. Parameter").

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

nur bei INIT PU: Die von openUTM übertragenen Informationen maximal in der in KCLI angegebenen Länge.

KB-Kopfbereich

im KB-Kopfbereich die aktuellen Daten des KB-Kopfes (siehe Tabelle).

KCRLM

nur bei INIT PU: im Feld KCRLM die tatsächliche Länge der bei UTM vorhandenen Information. Falls im Feld KCLI ein kleinerer Wert angegeben wurde, wird die Information in der in KCLI angegebenen Länge zurückgegeben und KCRCCC auf 07Z gesetzt. Bei KCRCCC >= 40Z werden keine Informationen übertragen, deshalb ist in diesen Fällen KCRLM=0.

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

nur bei INIT oder INIT PU:

- Bei einer Nachricht von einem Terminal:

  Leerzeichen (im Zeilenmodus) oder den Formatnamen (im Formatmodus) der letzten Bildschirmausgabe, also den Namen der beim MPUT des letzten Dialog-Schritts im Feld KCMF/kcfn angegeben wurde. Bestand die letzte Ausgabe aus mehreren Teilformaten, so enthält KCRMF/kcrfn den Namen des ersten Teilformats, in das Daten eingegeben wurden. Wurden in keines der Teilformate Daten eingegeben, so enthält KCRMF/kcrfn den Namen des ersten Teilformats.

  Nur auf BS2000-Systemen: Wurde bei der letzten Bildschirmausgabe ein Editprofil verwendet, so enthält KCRMF/kcrfn dieses Editprofil.
- Bei einer Nachricht von einer TS-Anwendung oder einem Teilprogramm der gleichen Anwendung: Leerzeichen.
- Wenn eine Rücksetznachricht vorliegt (nach PEND RS): Leerzeichen.

- bei einer Nachricht vom LU6.1-Partner oder UPIC-Client:

  Das Formatkennzeichen des ersten Nachrichtenteils, das der LU6.1-Partner oder UPIC-Client beim Senden angegeben hat.

  Besonderheit im LU6.1-Auftraggeber-Vorgang:

  Leerzeichen, falls für die in KCRPI angegebene Vorgangs-Id eine Statusinformation vorliegt.

- bei verteilter Verarbeitung über OSI TP:

  Wurde der Teilprogrammlauf auf Grund eines verteilten Dialogs gestartet, so enthält KCRMF/kcrfn im Auftraggeber-Vorgang den Namen der abstrakten Syntax, die der Nachricht vom Auftraggeber zugeordnet wurde; enthält das Feld Leerzeichen, dann ist als abstrakte Syntax die UDT-Syntax ausgewählt.

  Im Auftragnehmer-Vorgang enthält KCRMF/kcrfn den Namen der abstrakten Syntax, die der Nachricht von dem in KCRPI beschriebenen AN-Vorgang zugeordnet wurde; enthält das Feld Leerzeichen, dann ist als abstrakte Syntax die UDT-Syntax ausgewählt, oder von dem Partner liegt eine Fehlernachricht vor.

KCRPI

nur bei INIT oder INIT PU:

- Bei einer Nachricht von einem UPIC-Client-Programm, Terminal, einer TS-Anwendung oder einem Teilprogramm der gleichen Anwendung: Leerzeichen.
- Wenn eine Rücksetznachricht vorliegt: die Reset-Id.
- Im Auftraggeber-Vorgang bei verteilter Verarbeitung:
  die Vorgangs-Id des Auftragnehmer-Vorgangs, wenn eine Nachricht vom Auftragnehmer vorliegt.
- im Auftragnehmer-Vorgang bei verteilter Verarbeitung: Leerzeichen.

KCKBPRG/kclcapa

in KCKBPRG/kclcapa (KB-Programmbereich) die Daten des Vorgangs, falls der Programmlauf Vorgangs-spezifische Daten von einem anderen Programm übernimmt. Ist es der erste Teilprogrammlauf eines Vorgangs, so ist der Bereich undefiniert bzw. mit dem generierten Füllzeichen belegt. Wurde im Vorgängerteilprogramm in KCLKBPRG/kclcapa die Länge Null angegeben, so werden keine Daten an den Folge-Teilprogrammlauf übergeben.

KDCS-Returncodes im Feld KCRCCC beim INIT-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde durchgeführt.
Bei INIT PU: die angeforderten Informationen wurden in voller Länge in den Nachrichtenbereich übertragen.

01Z Die in KCLKBPRG/kclcapa angegebene Länge überschreitet den bei der Generierung der Anwendung angegebenen Wert.

02Z Bei INIT oder INIT PU:
Die in KCLPAB/kclspa angegebene Länge überschreitet den bei der Generierung der Anwendung angegebenen Wert.

07Z Bei INIT PU:
Die Funktion wurde ausgeführt, der bereitgestellte Nachrichtenbereich ist zu kurz (Länge in KCLI zu kurz).
Es wurde keine oder nur unvollständige Informationen zurückgegeben.

48Z Bei INIT PU:
Ungültige Version der Datenstruktur.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z Bei INIT oder INIT PU: Der INIT-Aufruf wurde in diesem Programmlauf bereits gegeben.

73Z Eine Längenangabe ist negativ. Bei INIT PU: KCLI ist ungültig.

77Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

88Z Ungültige Schnittstellenversion.

89Z Bei INIT PU oder INIT MD:
Beim Aufruf der Funktion waren nicht verwendete Parameter nicht mit binär null versorgt.

Rückgaben von openUTM im Kopf des KDCS-Kommunikationsbereichs

Unter anderem gibt openUTM nach dem INIT-Aufruf zurück: den TAC des Vorgangsbeginns, den aktueller TAC, Datum, Uhrzeit und den LTERM-Partner des Absenders.
Im Feld KCLKBPB/kclpa steht die generierte Länge des KB-Programmbereichs. Welche Angaben openUTM im KB-Kopf zurückgibt, zeigt die folgende Tabelle:

Feldname COBOL	Feldname C/C++	Inhalt (von openUTM eingetragen)
KCBENID	kcuserid	UTM-Benutzerkennung, unter der der Service initiiert wurde. Bei Betrieb ohne Benutzerkennungen: KCBENID/kcuserid= KCLOGTER. Bei verteilter Verarbeitung über LU6.1: lokaler Sessionnamen (LSES). Bei verteilter Verarbeitung über OSI TP und Security-Typ "N": lokaler Name der Verbindung (OSI-TP Association), sonst Benutzerkennung.
Vorgangs-spezifische Daten:		
KCTACVG	kccv_tac	TAC, der verwendet wurde, um diesen Vorgang zu starten
KCTAGVG	kccv_day	Tag des Vorgangsbeginns
KCMONVG	kccv_month	Monat des Vorgangsbeginns
KCJHRVG	kccv_year	Jahr des Vorgangsbeginns
KCTJHVG	kccv_doy	Industrietag des Vorgangsbeginns
KCSTDVG	kccv_hour	Stunde des Vorgangsbeginns
KCMINVG	kccv_minute	Minute des Vorgangsbeginns
KCSEKVG	kccv_second	Sekunde des Vorgangsbeginns

Feldname COBOL	Feldname C/C++	Inhalt (von openUTM eingetragen)
KCKNZVG	kccv_status	Vorgangskennzeichen • F: erster Teilprogrammlauf eines Dialog-Vorgangs • A: erster Teilprogrammlauf eines Asynchron-Vorgangs • N: Folge-Teilprogrammlauf eines Vorgangs • C: erster Teilprogrammlauf eines geketteten Vorgangs • R: Wiederanlauf eines Vorgangs • D: Vorgangsende durch Verbindungsverlust (nur bei LTERM-Partnern, die mit RESTART=NO generiert wurden) • Z: Vorgangsende durch Abbruch • E: normales Vorgangsende • L: Ende des letzten Prozesses bei normaler Beendigung
		Die Vorgangskennzeichen D, Z, E können nur im Exit VORGANG auftreten, das Vorgangskennzeichen L tritt nur im Exit SHUT auf. Alle anderen Vorgangskennzeichen können sowohl im KDCS-Aufruf INIT als auch in Exit VORGANG auftreten.
Teilprogramm-spezifische Daten:		
KCTACAL	kcpr_tac	TAC, mit dem das Programm adressiert wurde
KCSTDAL	kcpr_hour	Stunde des Teilprogrammbeginns
KCMINAL	kcpr_minute	Minute des Teilprogrammbeginns
KCSEKAL	kcpr_second	Sekunde des Teilprogrammbeginns
KCAUSWEIS	kccard	Ausweis-Kennzeichen: A (Ausweis steckt) oder Leerzeichen
KCTAIND	kctaind	Transaktionskennzeichen: F (erste) oder N (Folgetransaktion)
KCLOGTER	kclogter	LTERM-Name; bei verteilter Verarbeitung: (OSI-)LPAP-Name
KCTERMN	kctermn	Kennzeichen des Kommunikationspartners, bei verteilter Verarbeitung über LU6.1: CON....,TERMN=, bei verteilter Verarbeitung über OSI TP: OSI-LPAP....,TERMN= sonst: PTERM ..., TERMN (siehe auch Tabelle bei PTERM im openUTM-Handbuch „Anwendungen generieren“).
KCLKBPB	kclpa	Maximallänge des KB-Programmbereichs, wie sie bei der Generierung festgelegt wurde.

Feldname COBOL	Feldname C/C++	Inhalt (von openUTM eingetragen)
Daten bezüglich Vorgangs-Kellerung:		
KCHSTA	kchsta	Stapelhöhe, d.h. Anzahl der gekellerten Vorgänge aus der Sicht des aktuellen Vorgangs (0 bis 15).
KCDSTA	kcdsta	Veränderung der Stapelhöhe: + (Zunahme), - (Abnahme) oder 0 (unverändert, auch bei erneuter Kellerung nach Rückkehr aus einem eingeschobenen Vorgang)
KCPRIND	kcprind	Programmindikator: A = Asynchron-Vorgang, D = Dialog-Vorgang
KCOF1	kcof1	OSI TP-Funktionen in einem OSI TP-Auftragnehmer-Vorgang
KCCP	kccp	Indikator für das Client-Protokoll: • 0: Asynchron-Verarbeitung • 1: LU6.1 • 2: OSI TP • 3: UPIC • 4: DTP (Unix-, Linux- und Windows-Systeme) • 4: TIAM (BS2000-Systeme) • 5: APPLI • 6: SOCKET (USP) • 7: HTTP • 8: USP-SECURE • 9: HTTPS
KCTARB	kctarb	Information über Rücksetzen einer OSI TP-Transaktion
KCYEARVG	kccv_year4	Jahr des Vorgangsstarts (vierstellig)

Eigenschaften des KB-Programmbereichs und des SPAB

- Der KB-Programmbereich ist einem Vorgang zugeordnet, der SPAB einem Teilprogrammlauf.
- Bei Vorgangsbeginn ist der Inhalt des KB-Programmbereichs und des SPAB undefiniert bzw. mit dem generierten Füllzeichen belegt.
  Ein solches Füllzeichen kann z.B. die Fehlersuche beim Test erleichtern oder zu Datenschutzzwecken eingesetzt werden. Mit diesem Zeichen werden SPAB und KB-Programmbereich beim Start eines Prozesses vorbelegt und am Ende eines Verarbeitungsschritts überschrieben, siehe openUTM-Handbuch „Anwendungen generieren“.
- Wurde beim INIT eines Teilprogramms ein KB-Programmbereich mit einer Länge von *n* Bytes angegeben und wird im darauffolgenden Teilprogrammlauf ein größerer KB-Programmbereich von *m* Bytes (*m* > *n*) angefordert, so sind die letzten (*m*-*n*)-Bytes des KB-Programmbereichs ebenfalls undefiniert bzw. mit dem generierten Füllzeichen belegt.

Besonderheiten des INIT-Aufrufs bei verteilter Verarbeitung

- INIT-Aufruf im Auftraggeber-Vorgang

  Wurde die verteilte Transaktion zurückgesetzt, dann gibt openUTM beim INIT des ersten Teilprogramms einer Folgetransaktion das Vorgangskennzeichen "R" zurück (im Feld KCKNZVG/kccv_status des KB-Kopfes). In diesem Fall liegt meist eine Statusinformation vom Auftragnehmer-Vorgang vor.

  In dem Folge-Teilprogramm liefert der INIT-Aufruf im Auftraggeber-Vorgang zusätzlich folgende Informationen im KDCS-Rückgabebereich:

  - KCRPI enthält die Vorgangs-Id des Auftragnehmer-Vorgangs, der dieses Teilprogramm gestartet hat.
  - KCRMF/kcrfn enthält das Formatkennzeichen, das der Auftragnehmer-Vorgang im ersten Nachrichtenteil an den Auftraggeber-Vorgang eingetragen hat, sonst Leerzeichen.

  Der erste MGET-Aufruf zum Lesen des Ergebnisses muss mit KCRN=KCRPI und KCMF=KCRMF (bzw.: kcfn=kcrfn) erfolgen.

- INIT-Aufruf im Auftragnehmer-Vorgang

  Bei den Einträgen im KB-Kopf ergeben sich folgende Änderungen:

  KCBENID/kcuserid

  Bei LU6.1-Protokoll: enthält den lokalen Sessionnamen (LSES-Name, siehe LSES-Anweisung bei KDCDEF).
  Bei OSI TP-Protokoll mit Security-Typ "N": lokaler Name der Association (siehe Operand ASSOCIATION-NAMES in der KDCDEF-Anweisung OSI-LPAP), sonst Benutzerkennung.

  KCAUSWEIS/kccard

  enthält Leerzeichen, d.h. Ausweisleser wird nicht unterstützt.

  KCLOGTER

  enthält den logischen Namen der Partner-Anwendung (LPAP-Name bzw. OSI-LPAP-Name, siehe LPAP-Anweisung bei LU6.1-Protokoll, bzw. OSI-LPAP-Anweisung bei OSI TP-Protokoll: bei KDCDEF).

  KCTERMN

  enthält das Kennzeichen (**Ter**minal **Mn**emonic) der Partner-Anwendung, (siehe Operand TERMN= der CON-Anweisung bei LU6.1-Protokoll, bzw. der OSI-LPAP-Anweisung bei OSI TP-Protokoll: bei KDCDEF).

  KCOF1

  zeigt in einem OSI TP AN-Vorgang die OSI TP-Funktionen an, die für den Dialog mit dem AG ausgewählt wurden. Folgende Werte sind möglich:

Leerzeichen	der aktuelle Vorgang ist kein Auftragnehmer-Vorgang, oder zur Kommunikation mit dem AG-Vorgang wird nicht das OSI TP-Protokoll verwendet
B	Basisfunktionen
H	Basis- und Handshake-Funktionen
C	Basis- und Commit-Funktionen mit Chained Transactions
O	(other combination) für den Dialog mit dem AG wurde keine Standardkombination von OSI TP-Funktionen ausgewählt. Die ausgewählten OSI TP-Funktionen sind nur mit einem INIT PU-Aufruf lesbar.

KCCP (**c**lient **p**rotocol)

zeigt das für die Kommunikation verwendete Protokoll:

1 LU.1

2 OSI TP

KCTARB

zeigt in einem OSI TP-Vorgang an, ob bei einem früheren PGWT-Aufruf eine Situation eingetreten war, die ein Rücksetzen der Transaktion erfordert:

Leerzeichen	es ist keine Situation eingetreten, die ein Rücksetzen der Transaktion erfordert.
Y	bei einem früheren PGWT-Aufruf ist eine Situation eingetreten, die ein Vorsetzen der Transaktion nicht erlaubt, und die Transaktion wurde bisher nicht zurückgesetzt. Es ist noch Kommunikation mit den Partner-Vorgängen erlaubt. Ein Aufruf zum Vorsetzen der Transaktion führt zum abnormalen Vorgangsende.

Im KB-Rückgabebereich liefert das Feld KCRMF/kcrfn Informationen über den Partner-Vorgang.

Rückgaben im Nachrichtenbereich bei INIT PU

Wird der INIT-Aufruf mit der Operationsmodifikation PU verwendet, dann stellt openUTM dem Teilprogramm zusätzlich im Nachrichtenbereich Informationen über Anwendung, System und Kommunikationspartner zur Verfügung.

Für die Strukturierung des Nachrichtenbereichs stehen Ihnen Sprach-spezifische Datenstrukturen zur Verfügung. Für COBOL sind diese im COPY-Element KCINIC definiert, für C/C++ in der Include-Datei *kcini.h.* Im Kopf der Datenstruktur müssen Sie angeben, welche Informationen openUTM zurückliefern soll.

Folgende Informationen stellt openUTM zur Verfügung:

- die generierten Längen für den KB-Programmbereich und den Standard Primären Arbeitsbereich. Diese Information liefert der INIT PU-Aufruf in jeder Variante zurück**.**

- Datum und Uhrzeit des Anwendungsstarts und des Starts des Teilprogrammlaufs
- Informationen zu Anwendung und System
- Informationen über den Kommunikationspartner:

  In einem Dialog-Vorgang sind das Informationen über:

  - Namen des Kommunikationspartners,
  - Rechnernamen des Kommunikationspartners,
  - Namen der UTM-Anwendung, über den die Verbindung zum Kommunikationspartner aufgebaut wurde (BCAMAPPL-Name)

  In einem Asynchron-Vorgang werden stattdessen Leerzeichen übergeben.

- Informationen über die Sprachumgebung der Benutzerkennung, unter der der Vorgang gestartet wurde.

  Unix-, Linux- und Windows-Systeme:

  Diese Information umfasst Sprach- und Territorialkennzeichen sowie $LANG-Variable. Sprach- und Territorialkennzeichen werden zur Laufzeit des Programms aus der $LANG-Variablen gewonnen. Beispiel: Aus $LANG=En_US.ASCII erzeugt openUTM das Sprachkennzeichen En und das Territorialkennzeichen US.

  In einem Asynchron-Vorgang wird die Sprachumgebung des Benutzers übergeben, der den Vorgang gestartet hat.

  BS2000-Systeme:

  Diese Information umfasst:

  - Sprach- und Territorialkennzeichen sowie den Zeichensatz des Benutzers. Ist noch kein Benutzer angemeldet, übergibt openUTM Sprach- und Territorialkennzeichen sowie den Zeichensatz des LTERM-Partners.
  - den Namen des Zeichensatzes der Nachricht
  - die Information, ob der Benutzer an einem 7- oder 8-Bit-Terminal konnektiert ist.

  In einem lokal gestarteten Asynchron-Vorgang wird das Locale des Benutzers übergeben, der den Vorgang gestartet hat. Die Information über die 8-Bit-Fähigkeit des Terminals enthält in einem solchen Asynchron-Vorgang den Wert "7" und der Zeichensatzname der Nachricht enthält Leerzeichen.
- Informationen über den Auftraggeber-Vorgang bei der Kommunikation über OSI TP.
- Informationen über Verschlüsselungsverfahren, die zwischen der UTM-Anwendung und dem Client verwendet werden.
- Verschiedene Informationen, z.B. die Anzahl der Asynchron-Nachrichten in der Queue des Benutzers, die Gültigkeit des Passwortes, Zeitpunkt der letzten Anmeldung des Benutzers, Eigenschaften des LTERM und OSI-LPAP-Partner, von dem der Vorgang gestartet wurde, bezüglich LTERM-Gruppen und LTERM-/LPAP-Bündeln.
- Informationen über Asynchron-Nachrichten für den Benutzer.

Struktur des Nachrichtenbereichs bei INIT PU (gemäß KCINIC bzw. kcini.h)

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Header: Versionsnummer und angeforderte Informationen			
KCVER	if_ver	2	vor dem Aufruf zu versorgen: Versionsnummer der Datenstruktur (7)
KCDATE	dattim_info	1	vor dem Aufruf zu versorgen: Anfordern von Datum und Uhrzeit (Y/N)
KCAPPL	appl_info	1	vor dem Aufruf zu versorgen: Anfordern von Anwendungsinformation (Y/N)
KCLOCALE	locale_info	1	vor dem Aufruf zu versorgen: Anfordern von Locale Information (Y/N)
KCOSITP	ositp_info	1	vor dem Aufruf zu versorgen: Anfordern von OSI TP Informationen (Y/N)
KCENCR	encr_info	1	vor dem Aufruf zu versorgen: Anfordern von Verschlüsselungs-Informationen (Y/N)
KCMISC	misc_info	1	vor dem Aufruf zu versorgen: Anfordern von verschiedene Informationen (Y/N)
KCHTTP	http_info	1	vor dem Aufruf zu versorgen: Anfordern von HTTP-Informationen (Y/N)
		7	Reserviert für spätere Erweiterungen
Allgemeine Informationen, die immer zurückgegeben werden			
KCGPAB	gen_spab_lth	2	generierte Länge des SPAB
KCGNB	gen_nb_lth	2	generierte Länge des Nachrichtenbereichs
Informationen über Datum und Uhrzeit des Starts der Anwendung und des Teilprogrammlaufs (nur bei KCDATE/dattim_info=Y)			
KCADAY	as_dt_day	2	Tag des Anwendungsstarts
KCAMONTH	as_dt_month	2	Monat des Anwendungsstarts
KCAYEAR	as_dt_year	4	Jahr des Anwendungsstarts
KCADOY	as_dt_doy	3	Tag des Jahres des Anwendungsstarts
KCAHOUR	as_tm_hour	2	Stunde des Anwendungsstarts

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Informationen über Datum und Uhrzeit des Starts der Anwendung und des Teilprogrammlaufs (nur bei KCDATE/dattim_info=Y)			
KCAMIN	as_tm_minute	2	Minute des Anwendungsstarts
KCASEC	as_tm_second	2	Sekunde des Anwendungsstarts
KCASEAS	as_season	1	Zeit des Anwendungsstarts ist in Normalzeit (W) oder Sommerzeit (S) angegeben, stellt das Betriebssystem keine Informationen über Sommer-/Normalzeit zur Verfügung, werden Leerzeichen ausgegeben.
KCPDAY	ps_dt_day	2	Tag des Programmstarts
KCPMONTH	ps_dt_month	2	Monat des Programmstarts
KCPYEAR	ps_dt_year	4	Jahr des Programmstarts
KCPDOY	ps_dt_doy	3	Tag des Jahres des Programmstarts
KCPHOUR	ps_tm_hour	2	Stunde des Programmstarts
KCPMIN	ps_tm_minute	2	Minute des Programmstarts
KCPSEC	ps_tm_second	2	Sekunde des Programmstarts
KCPSEAS	ps_season	1	Zeit des Programmstarts ist in Normalzeit (W) oder Sommerzeit (S) angegeben, stellt das Betriebssystem keine Informationen über Sommer-/Normalzeit zur Verfügung, werden Leerzeichen ausgegeben.
KCTMZONE	time_zone	12	*Unix-, Linux- und Windows-Systeme:* Leerzeichen
			BS2000-Systeme: Zeitzone in der Form *sHH:MM-hh:mm*. Dabei bedeuten: s "+" oder "-": Vorzeichen des Zeitunterschieds der lokalen Zeitzone zur UTC (Universal Time Coordinate, entspricht Greenwich-Zeit). HH:MM Zeitunterschied der lokalen Zeitzone zur UTC in Stunden (HH) und Minuten (MM). hh:mm Zeitverschiebung in der lokalen Zeitzone zwischen Sommer- und Normalzeit in Stunden (hh) und Minuten (mm).

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Information zu Anwendung, System und Kommunikationspartner (nur bei KCAPPL/appl_info=Y)			
KCAPPLNM	applnm	8	Name der UTM-Anwendung.
KCHOSTNM	hostm	8	*BS2000-Systeme:* Name des Rechners, an dem die Anwendung abläuft
KCHOSTNM	hostm	8	*Unix-, Linux- und Windows-Systeme:* • Stand-alone-Anwendungen: Name des mit MAX HOSTNAME generierten Hosts. • UTM-Cluster-Anwendungen: Name des mit CLUSTER-NODE VIRTUAL-HOST generierten Hosts.
KCPTRMNM	ptrmnm	8	Bei einem Dialog-Vorgang: • bei verteilter Verarbeitung über LU6.1: CON-Name des Kommunikationspartners, • bei verteilter Verarbeitung über OSI TP: OSI-CON-Name des Kommunikationspartners, • sonst PTERM-Name des Kommunikationspartners. Bei einem Asynchron-Vorgang: Leerzeichen.
KCPRONM	pronm	8	Bei einem Dialog-Vorgang: • im OSI TP-Auftragnehmer-Vorgang: Leerzeichen, • sonst: Rechnername des Kommunikationspartners. Bei einem Asynchron-Vorgang: Leerzeichen.
KCBCAPNM	bcapnm	8	Bei einem Dialog-Vorgang: • OSI TP-Auftragnehmer-Vorgang: ACCESS-POINT-Name, über den die Association zum Partner aufgebaut ist, • sonst: BCAMAPPL-Name, über den die Verbindung zum Partner aufgebaut ist. Bei einem Asynchron-Vorgang: Leerzeichen
KCVERS	version	6	openUTM-Version V*nn.nx* (z.B. V07.0A).
KCIVER	iversion	2	Versionsnummer der KDCS-Schnittstelle, zeigt den Stand der verfügbaren Funktionserweiterungen der KDCS-Schnittstelle an; in openUTM V7.0 den Wert 9.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Information zu Anwendung, System und Kommunikationspartner (nur bei KCAPPL/appl_info=Y)			
KCIVAR	ivariant	1	Kennzeichen für die Produktvariante von openUTM: 'B' für BS2000-Systeme 'X' für Unix- und Linux-Systeme 'N' für Windows-Systeme.
KCHSTNML	hostnm_long	64	*BS2000-Systeme:* • langer Name des Rechners, auf dem die Anwendung abläuft.
			Unix-, Linux- und Windows-Systeme: • Stand-alone-Anwendungen: langer Name des mit MAX HOSTNAME generierten Hosts. • UTM-Cluster-Anwendungen: langer Name des mit CLUSTER-NODE VIRTUAL-HOST generierten Hosts.
KCPRONML	pronm_long	64	Bei einem Dialog-Vorgang: • OSI-TP-Auftragnehmer-Vorgang: Leerzeichen, • sonst: langer Rechnername des Kommunikationspartners. Bei einem Asynchron-Vorgang: Leerzeichen.
Information über das Locale der Benutzerkennung, die den Vorgang gestartet hat (nur bei KCLOCALE/locale_info=Y).			
KCUSLANG	us_lang_id	2	Sprachkennzeichen des Benutzers; ist noch kein Benutzer angemeldet: Sprachkennzeichen des LTERM-Partners.
KCUSTERR	us_terr_id	2	Territorialkennzeichen des Benutzers; ist noch kein Benutzer angemeldet:Territorialkennzeichen des LTERM-Partners.
Nur auf Unix-, Linux- und Windows-Systemen:			
KCUSNLSL	us_nlslang	16	$LANG-Variable des Benutzers.
		10	Leerzeichen.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Information über das Locale der Benutzerkennung, die den Vorgang gestartet hat (nur bei KCLOCALE/locale_info=Y).			
Nur auf BS2000-Systemen:			
KCUSCCSN	us_ccsname	8	Zeichensatz des Benutzers; ist noch kein Benutzer angemeldet: Zeichensatz des LTERM-Partners.
		8	Leerzeichen.
KCCSCURR	curr_ccs	8	Zeichensatz der vom Terminal erhaltenen Nachricht (CCSN des am Terminal aktiven Zeichensatzes).
KCDEVCAP	dev_cap	1	Information, ob das Terminal ein 7- oder 8-Bit-fähiges Gerät ist. Die Angabe erfolgt in der Form "7" oder "8" (abdruckbar).

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
OSI TP-Informationen (nur bei KCOSITP/ositp_info=Y)			
KCFUPOL	fupol	1	Anzeige, ob die Functional Unit "Polarized Control" ausgewählt ist (Y/N).
KCFUHSH	fuhsh	1	Anzeige, ob die Functional Unit "Handshake" ausgewählt ist (Y/N).
KCFUCOM	fucom	1	Anzeige, ob die Functional Unit "Commit" ausgewählt ist (Y/N).
KCFUCHN	fuchn	1	Anzeige, ob die Functional Unit "Chained Transactions" ausgewählt ist (Y/N).
KCENDTA	endta	1	Dieses Feld zeigt an, ob am Ende des aktuellen Verarbeitungsschritts ein Transaktionsende angefordert werden darf und mit welchen Aufrufen dies gegebenenfalls geschehen muss. Werden in dem Verarbeitungsschritt nur Nachrichten an Auftragnehmer-Vorgänge gesendet, dann darf die Transaktion über das Ende des Verarbeitungsschritts hinaus offen bleiben. • Leerzeichen: keine Vorschrift bzgl. der Beendigung des Verarbeitungsschritts. • O: am Ende des Verarbeitungsschritts darf kein Transaktionsende anfordert werden. • R: die Transaktion und der Dialog-Schritt müssen abgeschlossen werden, der Vorgang darf nicht beendet werden. • S: die Transaktion muss abgeschlossen werden, der Dialog-Schritt darf nicht beendet werden. • C: die Transaktion muss abgeschlossen werden, der Vorgang darf nicht beendet werden. • F: die Transaktion muss abgeschlossen und der Vorgang beendet werden.
KCSEND	send	1	Dieses Feld zeigt an, ob in dem Verarbeitungsschritt eine Nachricht an den AG-Vorgang gesendet werden darf. • Y: Das Senden einer Nachricht an den Auftraggeber ist am Ende des Dialog-Schritts notwendig. Das Senden einer Nachricht an den Auftraggeber ist am Ende der Transaktion notwendig, wenn KCENDTA "S" enthält. • N: ein MPUT an den AG ist nicht möglich; an AN-Vorgänge dürfen Nachrichten gesendet werden, in diesem Fall muss jedoch die Transaktion über den Verarbeitungsschritt hinaus offen bleiben.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Verschlüsselungs-Informationen (nur bei KCENCR/encr_info=Y)			
KCPTERM	pterm_enclev	1	Generierte minimale Verschüsselungsebene des Clients in der zugehörigen PTERM- oder TPOOL-Anweisung: • N: Es ist keine minimale Verschlüsselungsebene für den Client generiert. • 3: Es ist die minimale Verschlüsselungsebene 3 für den Client generiert. • 4: Es ist die minimale Verschlüsselungsebene 4 für den Client generiert. • 5: Es ist die minimale Verschlüsselungsebene 5 für den Client generiert. • T: Es ist die minimale Verschlüsselungsebene TRUSTED für den Client generiert, der Client ist vertrauenswürdig.
KCCLIENT	client_enclev	1	Maximaler vom Client unterstützter Verschüsselungsmechanismus: • N: Es wird kein Verschüsselungsmechanismus vom Client unterstützt. • 3: Es wird maximal ein Verschüsselungsmechanismus der Ebene 3 unterstützt. • 4: Es wird maximal ein Verschüsselungsmechanismus der Ebene 4 unterstützt. • 5: Es wird maximal ein Verschüsselungsmechanismus der Ebene 5 unterstützt.
KCSESS	session_enclev	1	Auf der aktuellen Session zwischen Client un Server vereinbarter Verschüsselungsmechanismus: • N: Es wurde kein Verschüsselungsmechanismus vereinbart. • 3: Ein Verschüsselungsmechanismus der Ebene 3 wurde vereinbart. • 4: Ein Verschüsselungsmechanismus der Ebene 4 wurde vereinbart. • 5: Ein Verschüsselungsmechanismus der Ebene 5 wurde vereinbart.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Verschlüsselungs-Informationen (nur bei KCENCR/encr_info=Y)			
KCCNVTAC	convtac_enclev	1	Generierte minimale Verschlüsselungsebene der TACs, mit dem der Vorgang gestartet wurde. Dieses Feld ist auch im Asynchron-Vorgang versorgt: • N: Es ist keine minimale Verschlüsselungsebene für den TAC generiert. • 1: Es ist die minimale Verschlüsselungsebene 1 für den TAC generiert. • 2: Es ist die minimale Verschlüsselungsebene 2 für den TAC generiert. • 5: Es ist die minimale Verschlüsselungsebene 5 für den TAC generiert.
KCCONV	conv_enclev	1	Dieses Feld zeigt an, ob für den Vorgang eine Verschlüsselung vereinbart wurde. Dieser Wert wird durch den UPIC-Client ausgewählt oder durch die Verschlüsselungsebene des Transaktionscodes des Vorgangs bestimmt, nicht durch die der Session. Dieses Feld ist auch im Asynchron-Vorgang versorgt: • N: Für den Vorgang wurde keine Verschlüsselung vereinbart. • 3: Für den Vorgang wurde ein Verschlüsselungsmechanismus der Ebene 3 vereinbart. • 4: Für den Vorgang wurde ein Verschlüsselungsmechanismus der Ebene 4 vereinbart. • 5: Für den Vorgang wurde ein Verschlüsselungsmechanismus der Ebene 5 vereinbart.
KCINPMSG	inputmsg_enclev	1	Dieses Feld zeigt an, ob die Dialog-Eingabenachricht vom Client verschlüsselt war oder nicht: • N: Die Eingabenachricht war nicht verschlüsselt. • 3: Die Eingabenachricht war mit einem Verschlüsselungsmechanismus der Ebene 3 verschlüsselt. • 4: Die Eingabenachricht war mit einem Verschlüsselungsmechanismus der Ebene 4 verschlüsselt. • 5: Die Eingabenachricht war mit einem Verschlüsselungsmechanismus der Ebene 5 verschlüsselt.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
verschiedene Informationen über den Benutzer, der den Vorgang gestartet hat (KCBENID,kcuserid), das LTERM oder (OSI-)LPAP, von dem aus der Vorgang gestartet wurde (KCLOGTER,kclogter) und die Anwendung (nur bei KCMISC/misc_info=Y).			
KCUMSGS	amsgs_user	10	Anzahl der Asynchron-Nachrichten in der Queue des Benutzers.
KCPWVMAX	pw_val_max	2	Anzahl der Tage, die das Passwort des Benutzers noch gültig ist. Werte mit spezieller Bedeutung: • 0: Das Passwort wird innerhalb der nächsten 24h ungültig. • -1: Das Passwort ist ohne beschränkte Gültigkeitsdauer generiert. Das Passwort ist unbegrenzt gültig. • -2: Die Gültigkeit des Passwortes ist abgelaufen. Der Wert kann nur im Asynchron-Vorgang oder im Anmelde-Vorgang vorkommen. • -3: Die Passwort-Komplexität oder Mindestlänge des Passworts wurde erhöht und das mit dem Tool KDCUPD übertragene Passwort entspricht möglicherweise nicht den Anforderungen der generierten Komplexitätsstufe oder ist zu kurz. Der Wert kann nur im Asynchron-Vorgang oder im Anmelde-Vorgang vorkommen.
KCPWVMIN	pw_val_min	2	Anzahl der Tage, die das Passwort des Benutzers nur administrativ geändert werden kann, aber z.B. nicht über einen SIGN CP Aufruf (Mindestgültigkeitsdauer des Passwortes). Werte mit spezieller Bedeutung: • 0: Das Passwort darf geändert werden. • -1: Für den Benutzer darf kein Passwort gesetzt werden. Das kann in den folgenden Fällen auftreten: a) Anwendungen ohne Benutzer, b) Benutzer ist LU6.1-Session oder OSI TP Association, c) Verbindungs-User (nur bei bei TS- oder UPIC-Client), d) interner Benutzer KDCMSGUS, e) *BS2000-Systeme:* Benutzer, die mit Kerberos-Authentisierung generiert sind.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
verschiedene Informationen über den Benutzer, der den Vorgang gestartet hat (KCBENID,kcuserid), das LTERM oder (OSI-)LPAP, von dem aus der Vorgang gestartet wurde (KCLOGTER,kclogter) und die Anwendung (nur bei KCMISC/misc_info=Y).			
KCLSTSGN	last_sign	14	Datum und Uhrzeit der letzten Anmeldung. Datum und Uhrzeit werden in der Form YYYMMDDHHMMSS angegeben. In folgenden Fällen werden Nullen zurückgegeben: • im Anmelde-Vorgang nach dem ersten erfolgreichen Anmelden nach einer Neugenerierung, • interner Benutzer KDCMSGUS, • Benutzer ist LU6.1-Session oder OSI TP Association.
KCBNDLMS	bundle_master	8	Name des Masters des LTERM-/LPAP-Bündel, wenn das LTERM/ (OSI-)LPAP ein Slave dieses Bündel ist.
KCISGRMS	is_group_master	1	Das Feld zeigt an, ob das LTERM (oder der Master des LTERM-Bündel) der Master einer LTERM-Gruppe ist. • Y: (Master)-LTERM ist Gruppen-Master. • N: (Master)-LTERM ist kein Gruppen-Master.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
verschiedene Informationen über den Benutzer, der den Vorgang gestartet hat (KCBENID,kcuserid), das LTERM oder (OSI-)LPAP, von dem aus der Vorgang gestartet wurde (KCLOGTER,kclogter) und die Anwendung (nur bei KCMISC/misc_info=Y).			
KCLTCP	lterm_client_prot	1	Das Feld zeigt das Client-Protokoll des LTERM: • 0: Der Vorgang läuft für das interne LTERM KDCMSGLT (nur im Asynchron-Vorgang) • 1: LU6.1 • 2: OSI TP • 3: UPIC • 4: DTP (Unix-, Linux- und Windows-Systeme) • 4: TIAM (BS2000-Systeme) • 5: APPLI • 6: SOCKET (USP) • 7: HTTP • 8: USP-SECURE • 9: HTTPS
KCAPPLST	application_state	1	Das Feld zeigt den Status der Anwendung: • N: Die Anwendung läuft normal. • G: Die Anwendung ist im Zustand "Graceful Shutdown". • W: Die Anwendung ist im Zustand "Shutdown Warn". • S: Die Anwendung wird normal beendet. • T: Die Anwendung wird abnormal beendet.
KCKRBCAP	kerberos_capability	1	In einem Dialog-Vorgang zeigt das Feld an, ob der Client kerberos-fähig ist: • Y: Der Client ist kerberos-fähig. • N: Der Client ist nicht kerberos-fähig. In einem Asynchron-Vorgang: Leerzeichen.

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
verschiedene Informationen über den Benutzer, der den Vorgang gestartet hat (KCBENID,kcuserid), das LTERM oder (OSI-)LPAP, von dem aus der Vorgang gestartet wurde (KCLOGTER,kclogter) und die Anwendung (nur bei KCMISC/misc_info=Y).			
KCCDINFO	info_cd_available	1	*Nur auf BS2000-Systemen:* In einem Dialog-Vorgang ist folgende INFO CD Information verfügbar: • C: Ausweis-Information (Magnetstreifenkarte). • K: Kerberos-Information. • E: Kerberos-Information, aber der Kerberos-Dialog lieferte einen Fehler, oder die Information konnte nicht in ihrer gesamten Länge gespeichert werden, weil sie größer ist als der bei MAX PRINCIPAL-LTH generierte Wert. • N: Keine INFO CD Information. Im Asynchron-Vorgang ist INFO CD nicht erlaubt: Leerzeichen
HTTP-Informationen können nur in Dialog-Vorgängen versorgt werden, die von einem HTTP- oder HTTPS-Client aufgerufen wurden, d.h. im Feld KCCP im KBKOPF ist der Wert 7 oder 9 eingetragen). (nur bei KCHTTP/http_info=Y).			
KCHTMTD	httpMethod	1	HTTP-Methode: 1=GET, 2=PUT, 3=POST, 4=DELETE
KCHTVERS	httpVersion	1	HTTP-Version: 1=1.1
KCSCHEME	scheme	1	Schema: 1=HTTP, 2=HTTPS
KCHTEXIT	httpExit	1	Zeigt an, ob ein benutzerspezischer, der system-spezifische oder kein HTTP-Exit aufgerufen wird (U/S/N)
KCCDCONV	codeConversion	1	Code-Konvertierung der Ein- und Ausgabe-Nachricht: N keine Code-Konvertierung S System-Code-Konvertierung 1/2/3/4 Code-Konvertierung SYS1/2/3/4

7.13 LPUT Schreiben in die Protokolldatei

Mit dem Aufruf LPUT (logfile PUT) schreiben Sie einen Satz in die Benutzer-Protokolldatei. openUTM stellt jedem Satz den aktuellen Inhalt des KB-Kopfes voran. Die Maximallänge eines Satzes wird bei der Generierung festgelegt (MAX-Anweisung, Operand LPUTLTH).Die Reihenfolge der Sätze in der Protokolldatei entspricht nicht genau der zeitlichen Reihenfolge der LPUT-Aufrufe in der Anwendung.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**		
	KCOP	**KCLA**	**Nachrichtenbereich**
Schreiben in die Protokolldatei	"LPUT"	Länge	zu protokollierende Daten

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, aus dem openUTM die zu protokollierenden Daten lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"LPUT"
KCLA	Länge in Byte
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_LPUT	(nb,kcla)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den LPUT-Aufruf ein:

KCOP

im Feld KCOP den Operationscode LPUT.

KCLA

im Feld KCLA die Länge der zu übertragenden Daten in Bytes. Die Länge wird von openUTM vor den Datensatz geschrieben.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Daten ein, die Sie in die Benutzer-Protokolldatei schreiben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

Die Adresse des KDCS-Parameterbereichs.

2. Parameter

Die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs müssen Sie auch dann angeben, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächsten Abschnitt.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim LPUT-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
01Z	Der in KCLA angegebene Wert ist zu groß. Er wird auf die generierte Maximallänge gekürzt (LPUTLTH-Operand in der MAX-Anweisung).
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Timeout), siehe KCRCDC.
43Z	Die Längenangabe in KCLA ist ungültig (z.B. negativ).
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Programm wurde kein INIT gegeben.

Eigenschaften des LPUT-Aufrufs

- Vor dem nächsten Sicherungspunkt ist die LPUT-Operation noch rücksetzbar.
- Die Aufrufe PEND ER/FR/RS und RSET setzen die LPUT-Operation zurück.

Der Aufbau der Benutzer-Protokolldatei ist auf "Benutzer-Protokolldatei" beschrieben.

7.14 MCOM Definieren eines Auftrags-Komplexes

Mit dem Aufruf MCOM (message complex) können Sie:

- den Anfang eines Auftrags-Komplexes definieren und dabei die Ziele des Basisauftrags und seiner Quittungsaufträge festlegen, oder
- das Ende eines Auftrags-Komplexes definieren.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und ihre Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCRN	KCPOS	KCNEG	KCCOMID
Beginn des Auftrags-Komplexes	"MCOM"	"BC"	LTERM/TAC /Vorgangs-Id/TAC-Queue	TAC /Leerzeichen /TAC-Queue	TAC /Leerzeichen /TAC-Queue	Komplex-Id
Ende des Auftrags-Komplexes	"MCOM"	"EC"	binär null	binär null	binär null	Komplex-Id

Alle nicht verwendeten Felder des KDCS-Parameterbereichs müssen mit binär null versorgt werden.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"MCOM"
KCOM	"BC"/"EC"
KCRN	LTERM-Name/TAC/Vorgangs-Id/TAC-Queue/binär null
KCPOS	TAC/Leerzeichen/TAC-Queue/binär null
KCNEG	TAC/Leerzeichen/TAC-Queue/binär null
KCCOMID	Komplex-Id

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	-

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_MCOMBC	(kcrn,kcpos,kcneg,kccomid)
KDCS_MCOMEC	(kccomid)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den MCOM-Aufruf ein:

KCOP

im Feld KCOP den Operationscode MCOM.

KCOM

im Feld KCOM entweder "BC" für Beginn oder "EC" für Ende eines Auftrags-Komplexes.

KCRN

im Feld KCRN bei KCOM = BC

- den LTERM-Namen eines Kommunikationspartners, wenn der Basisauftrag ein Ausgabeauftrag ist,
- den TAC eines Asynchron-Programms, wenn der Basisauftrag ein Hintergrund-Auftrag (ohne verteilte Verarbeitung) ist oder
- den Namen der TAC-Queue, wenn der Basisauftrag ein Ausgabeauftrag in eine TAC-Queue (ohne verteilte Verarbeitung) ist.
- die Vorgangs-Id eines Auftragnehmer-Vorgangs, wenn der Basisauftrag an einen Auftragnehmer-Vorgang gerichtet ist.

Bei KCOM = EC muss binär null eingetragen werden.

KCPOS

im Feld KCPOS bei KCOM = BC als Ziel des positiven Quittungsauftrages den TAC eines Asynchron-Programms bzw. einer TAC-Queue oder Leerzeichen, falls kein positiver Quittungsauftrag erzeugt werden soll.

Bei KCOM = EC wird binär null eingetragen.

KCNEG

im Feld KCNEG bei KCOM = BC als Ziel des negativen Quittungsauftrages den TAC eines Asynchron-Programms bzw. einer TAC-Queue oder Leerzeichen, falls kein negativer Quittungsauftrag erzeugt werden soll.

Bei KCOM = EC wird binär null eingetragen.

KCCOMID

im Feld KCCOMID die Komplex-Identifikation (Komplex-Id) des Auftrags-Komplexes. Sie wird bei MCOM BC definiert, darf 2 bis 8 Zeichen lang sein und muss mit dem Zeichen "*" beginnen. Sie wird bei allen zum Komplex gehörenden DPUT-Aufrufen sowie bei MCOM EC angegeben.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim MCOM-Aufruf

Im Programm sind auswertbar:

000 Die Funktion wurde ausgeführt.

40Z openUTM kann die Funktion nicht durchführen: Generierungs- oder Systemfehler oder es sollte ein Auftrags-Komplex begonnen werden, ohne dass der vorhergehende Auftrags-Komplex abgeschlossen wurde.

42Z Der Eintrag in KCOM ist ungültig.

44Z Der Wert in KCRN ist ungültig:

- kein TAC eines Asynchron-Programms bzw. einer TAC-Queue angegeben oder TAC bzw. TAC-Queue gesperrt/verboten
- kein LTERM-Name oder LTERM-Name eines UPIC- oder HTTP-Clients angegeben
- keine gültige Vorgangs-Id angegeben oder die Vorgangs-Id wurde schon durch eine andere Nachricht belegt.
- die Dead Letter Queue (KDCDLETQ) darf nicht angegeben werden.

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

51Z Bei KCOM=EC fehlt zu einer Benutzerinformation der zugehörige (Quittungs-)Auftrag.

55Z Der Eintrag in KCCOMID ist ungültig: der Name beginnt nicht mit "*" oder er wurde im Teilprogramm schon einmal vergeben oder er ist nicht bekannt (bei MCOM EC).

57Z Der Wert in KCPOS ist ungültig:

- kein TAC eines Asynchron-Programms bzw. einer TAC-Queue angegeben oder TAC bzw. TAC-Queue gesperrt/verboten
- Angabe nicht gleich Leerzeichen.
- die Dead Letter Queue (KDCDLETQ) darf nicht angegeben werden.

58Z Der Wert in KCNEG ist ungültig:

- kein TAC eines Asynchron-Programms bzw. einer TAC-Queue angegeben oder TAC bzw. TAC-Queue gesperrt/verboten oder
- Angabe nicht gleich Leerzeichen.
- die Dead Letter Queue (KDCDLETQ) darf nicht angegeben werden.

Weitere Returncodes sind dem DUMP zu entnehmen:

71Z Im Teilprogramm wurde noch kein INIT gegeben.

Eigenschaften des MCOM-Aufrufs

- Ein Auftrags-Komplex muss vor dem PEND-Aufruf mit MCOM EC abgeschlossen werden, sonst bricht openUTM den Vorgang mit PEND ER und 86Z ab.
- Die Komplex-Identifikation muss innerhalb eines Teilprogramms eindeutig sein.
- MCOM BC ist erst dann erlaubt, nachdem alle Auftrags-Komplexe vorher abgeschlossen wurden.
- Wurde eine Vorgangs-Identifikation durch einen Auftrags-Komplex belegt, dann kann sie nur durch MCOM EC freigegeben werden (nicht wie sonst durch DPUT NE).
- Die Ziele von Quittungsaufträgen müssen Asynchron-Teilprogramme bzw. TAC-Queues der lokalen Anwendung sein.
- Ein Hauptauftrag eines Nachrichten-Komplexes mit negativem Quittungsauftrag wird im Fehlerfall (gegebenenfalls nach erfolgter Redelivery) nicht in der Dead Letter Queue gesichert, sondern gelöscht. Dafür wird der negative Quittungsauftrag aktiviert.

7.15 MGET Empfangen einer Dialog-Nachricht

Mit dem Aufruf MGET (message GET) können Sie in einem Teilprogrammlauf eines Dialog-Vorgangs Nachrichten in den Nachrichtenbereich einlesen. In einem Asynchron-Vorgang ist der MGET-Aufruf nur in Folge-Teilprogrammen oder in einem Folge-Verarbeitungsschritt zulässig.

Die Nachrichten können von den folgenden Absendern gesendet worden sein:

- von einem Terminal
- von einer anderen Anwendung (über LU6.1 oder OSI TP)
- von einer Transportsystem-Anwendung
- von einem HTTP-Client
- von einem UPIC-Client
- von einem vorhergehenden Teilprogrammlauf desselben Vorgangs

Bei Teilnachrichten muss jede Teilnachricht mit einem eigenen MGET gelesen werden.

Bei Socket-USP-Anwendungen und HTTP-Clients kann eine Teilnachricht mit mehreren MGET-Aufrufen gelesen werden. Anhand von KCRLM und des Returncodes kann festgestellt werden, ob eine Teilnachricht vollständig gelesen wurde.
Bei HTTP-Clients liest der erste MGET den Query-String, sofern ein solcher vorhanden ist. Enthält die Eingabenachricht keinen Query-String, dann gibt der erste MGET den Message-Body, bzw. den ersten Teil davon, zurück.

Stammt die Nachricht von einem OSI TP-Partner, so kann es sich bei der Nachricht um einen Nachrichtenteil, eine Fehlermeldung, eine Handshake-Anforderung oder eine Handshake-Quittung handeln.

Wurde eine Funktionstaste gedrückt, sind zwei MGET-Aufrufe erforderlich: der Erste liefert den Returncode, der Zweite die Daten.
Absender von Funktionstasten können nur Terminals oder UPIC-Clients sein.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich				
	KCOP	KCOM	KCLA	KCRN	KCMF/kcfn
BS2000-Systeme: Nachricht im Formatmodus	"MGET"	-	Länge	-	Formatkennzeichen
Nachricht im Zeilenmodus	"MGET"	-	Länge	-	Leerzeichen / Editprofil (nur auf BS2000-Systemen)
Nachricht vom vorhergehenden Teilprogramm derselben Anwendung	"MGET"	-	Länge	-	Leerzeichen
Rücksetznachricht von einem Teilprogramm	"MGET"	"NT"	Länge	Reset-Id	Leerzeichen
Dialog-Nachricht vom Auftraggeber-Vorgang	"MGET"	-	Länge	-	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax
Dialog-Nachricht vom Auftragnehmer-Vorgang	"MGET"	"NT"	Länge	Vorgangs-Id	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax
Statusinformation vom Auftragnehmer-Vorgang	"MGET"	"NT"	0	Vorgangs-Id	Leerzeichen

NT: Teilnachricht

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, in den openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"MGET"
KCOM	"NT"/-
KCLA	Länge in Byte
KCRN	Vorgangs-Id / Reset-Id / -
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Editprofil (nur auf BS2000-Systemen)

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_MGET	(nb,kcla,kcfn)
KDCS_MGETNT	(nb,kcla,kcrn,kcfn)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRDF	Bildschirmfunktion/0
KCRLM	tatsächliche Länge
KCRMGT	Art der Nachricht
KCVGST/kcpcv_state	Vorgangs-Status

KCTAST/kcpta_state	Transaktionsstatus
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Formatkennzeichen / Leerzeichen / Editprofil (nur auf BS2000-Systemen)
KCRPI	Vorgangs-Id/Leerzeichen

Im KDCS-Parameterbereich machen Sie für den MGET-Aufruf folgende Einträge:

KCOP

im Feld KCOP den Operationscode "MGET".

KCOM

nur anzugeben bei Nachrichten vom Auftragnehmer und bei Reset-Nachrichten: im Feld KCOM die Modifikation "NT" (Nachrichtenteil).

KCLA

im Feld KCLA die Länge, in der die Nachricht gelesen werden soll. Sie darf maximal so groß sein wie der Nachrichtenbereich, in den die Nachricht eingelesen werden soll (Länge null bedeutet: Kein Nachrichtenempfang; eine eventuell vorhandene Nachricht geht verloren). Die tatsächliche Länge der (Teil-) Nachricht wird im Feld KCRLM zurückgegeben.

KCRN

nur anzugeben bei Nachrichten vom Auftragnehmer und bei Reset-Nachrichten:

- bei einer Nachricht vom Auftragnehmer-Vorgang die Vorgangs-Id des Auftragnehmer-Vorgangs.
- bei einer Reset-Nachricht die Reset-Id der Rücksetznachricht.

> **i** Die Reset-Id bzw. die Vorgangs-Id können beim ersten MGET-Aufruf eines Teilprogramms aus dem Feld KCRPI des INIT-Aufrufs entnommen werden, bei weiteren MGET-Aufrufen aus dem Feld KCRPI des jeweils vorhergehenden MGET-Aufrufs.

KCMF/kcfn

im Feld KCMF/kcfn
(irrevant bei Nachrichten von vorhergehenden Teilprogrammläufen desselben Vorgangs)

- auf BS2000-Systemen bei Nachricht im Formatmodus:
  Formatkennzeichen
- bei Nachricht im Zeilenmodus:
  Leerzeichen
  auf BS2000-Systemen zusätzlich möglich:
  Editprofil

- beim Lesen einer Reset-Nachricht:
  Leerzeichen
- Nachricht von einem UPIC-Client:
  Formatkennzeichen, das der UPIC-Client beim Senden angegeben hat.
- bei verteilter Verarbeitung über LU6.1:
  Formatkennzeichen, das die Partner-Anwendung beim MPUT-Aufruf in KCMF/kcfn angegeben hat.
- bei Nachricht von OSI TP-Partner:
  Name der abstrakten Syntax der Nachricht. Dieser Name wurde beim vorausgegangenen INIT oder MGET-Aufruf im Feld KCRMF/kcrfn zurückgegeben. Leerzeichen stehen dabei für die abstrakte Syntax von UDT; in diesem Fall wird als Transfer Syntax BER verwendet, und die Decodierung der Nachricht übernimmt openUTM.
  Wird hier ein Wert ungleich Leerzeichen angegeben, dann übergibt openUTM die Nachricht an das Teilprogramm in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax; das Teilprogramm muss die Umsetzung in die lokale Darstellung selbst übernehmen. Dies kann z.B. mit Hilfe eines ASN.1-Compilers geschehen.

> **i** Dieses Feld ist auf alle Fälle korrekt versorgt, wenn Sie für den ersten MGET-Aufruf des Teilprogramms den beim INIT-Aufruf im Feld KCRMF/kcrfn zurückgegebenen Wert eintragen, bei weiteren MGET-Aufrufen den Wert aus dem Feld KCRMF/kcrfn des jeweils vorhergehenden MGET-Aufrufs.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   als 1. Parameter: die Adresse des KDCS-Parameterbereichs.

2. Parameter

   als 2. Parameter: die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht einlesen soll. Die Adresse des Nachrichtenbereichs müssen Sie auch dann angeben, wenn Sie in KCLM die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die (Teil-)Nachricht in der gewünschten Länge, falls nicht KCRCCC=03Z gesetzt wurde. War sie länger, als in KCLA angegeben, geht der Rest verloren.
Ausnahme: die Nachricht stammt von einer Socket-USP-Anwendung oder einem HTTP-Client. In diesem Fall wird der Returncode 02Z gesetzt und die restliche (Teil-)Nachricht kann mit dem nächsten MGET abgeholt werden.

KCRDF

im Feld KCRDF beim 1. MGET für einen Auftragnehmer-Vorgang, mit dem über das LU6.1-Protokoll kommuniziert wird, den Wert aus dem Feld KCDF des zugehörigen MPUT-Aufrufs im Auftragnehmer-Vorgang. In allen anderen Fällen enthält das Feld den Wert Null.

KCRLM

im Feld KCRLM die tatsächliche Länge der (Teil-)Nachricht. Falls im Feld KCLA ein kleinerer Wert angegeben wurde, wird die Nachricht in der in KCLA angegebenen Länge zurückgegeben. Bei Socket-USP-Anwendungen oder HTTP-Clients kann die restliche Nachricht noch gelesen werden (KCRCCC = 02Z), ansonsten geht die restliche Nachricht verloren (KCRCCC = 01Z).
Bei KCRCCC=03Z und bei leeren Nachrichten ist KCRLM=0.

Nur auf BS2000-Systemen: Bei Einsatz von FHS hängt der in KCRLM zurückgegebene Wert vom FHS-Startparameter KCRLM= ab.

KCRMGT

im Feld KCRMGT wird angezeigt was mit dem MGET gelesen wurde:

- "M" (message)
  eine Nachricht. Beim MGET für einen Partner, mit dem nicht über das OSI TP-Protokoll kommuniziert wird, ist nur der Wert "M" möglich.

Beim MGET für einen Partner, mit dem über das OSI TP-Protokoll kommuniziert wird, sind zusätzlich möglich:

- "C" (confirm)
  eine positive Handshake-Quittung,
- "E" (error)
  eine Fehlermeldung bzw. negative Handshake-Quittung,
- "H" (handshake)
  eine Handshake-Anforderung.

KCVGST/kcpcv_state

im Feld KCVGST/kcpcv_state den Vorgangs-Status des (Partner-)Vorgangs, siehe "MGET Empfangen einer Dialog-Nachricht".

KCTAST/kcpta_state

im Feld KCTAST/kcpta_state den Transaktionsstatus des Partner-Vorgangs, siehe "MGET Empfangen einer Dialog-Nachricht".

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe "MGET Empfangen einer Dialog-Nachricht".

KCRCDC

im Feld **KCRCDC** den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

im Feld KCRMF/kcrfn

- bei einer Nachricht im Zeilenmodus: Leerzeichen
  oder auf BS2000-Systemen: Name des Editprofils.

- bei einer Nachricht von einem Partner-Vorgang:
  KCRMF/kcrfn enthält das Formatkennzeichen bzw. den Namen der abstrakten Syntax der nächsten (Teil-) Nachricht, die von dem im Feld KCRPI benannten Vorgang gelesen werden kann. Nach dem MGET für die letzte (Teil-)Nachricht enthält dieses Feld das Formatkennzeichen der letzten (Teil-)Nachricht.

Nur auf BS2000-Systemen:

- nach dem Lesen eines ganzen Formats: Kennzeichen dieses Formats. Es ist immer gleich dem Kennzeichen des letzten Ausgabeformats.
- nach dem Lesen eines Teilformats: Kennzeichen des nächsten Teilformats mit Eingabedaten. Gibt es kein weiteres Teilformat mit Eingabe-Daten, so enthält KCRMF/kcrfn das Kennzeichen des zuletzt eingelesenen Teilformats. In diesem Fall ist KCRMF=KCMF (kcrfn=kcfn).

KCRPI

im Feld KCRPI

- bei einer Nachricht von einem Auftragnehmer-Vorgang:
  Vorgangs-Id eines Auftragnehmer-Vorgangs, von dem noch Nachrichtenteile oder Statusinformationen vorliegen, die noch nicht gelesen wurden.
- in allen anderen Fällen: Leerzeichen.

Vorgangs-Status im Feld KCVGST/kcpcv_state

Eintrag bei Dialogen ohne verteilte Verarbeitung:

- "O" (open)
  Der lokale Vorgang ist offen.

Einträge bei verteilter Verarbeitung nach einer Nachricht vom Partner-Vorgang:

- "C" (closed)
  Der Auftragnehmer-Vorgang hat sich beendet (PEND FI).
- "D" (disconnected)
  Die Kommunikation mit dem Auftragnehmer wurde wegen eines Verbindungsverlusts beendet (nur bei OSI TP).
- "E" (error)
  Nur bei Kommunikation über LU6.1:
  Der Auftragnehmer-Vorgang hat sich mit PEND ER oder PEND FR beendet.
- "I" (inactive)
  Der Auftragnehmer-Vorgang ist inaktiv, d.h. er konnte nicht gestartet werden, weil z.B. der Transaktionscode unbekannt oder gesperrt ist oder bei SOI TP keine Association belegt werden konnte.
- "O" (open)
  Der Partner-Vorgang ist offen, weitere Nachrichten können an ihn gesendet werden.
- "P" (pending end dialogue)
  Dieser Status kann nur bei heterogener Kopplung und bei Dialogen auftreten, für die die Commit Funktionalität nicht ausgewählt ist:
  Der Auftragnehmer-Vorgang möchte die Kommunikation beenden. Wenn der Auftraggeber-Vorgang nicht einverstanden ist, kann er die Kommunikation mit einem MPUT EM fortsetzen.

- "R" (reset)
  Nur bei Kommunikation über LU6.1:
  Der Auftragnehmer-Vorgang wurde mit PEND RS beendet.
- "T" (time out)
  Der Auftragnehmer-Vorgang wurde bzw. wird fehlerhaft beendet, da innerhalb der generierten Wartezeit keine Antwort vom Auftragnehmer-Vorgang eingetroffen ist oder innerhalb der generierten Wartezeit keine Session belegt werden konnte.
- "Z" (error)
  Der Auftragnehmer-Vorgang wurde vom System mit PEND ER beendet (z.B. KDCS-Aufruf im Auftragnehmer-Vorgang mit Fehleranzeige >= 70Z). Der Vorgangs-Status Z wird auch gesetzt, wenn ein Teilprogrammlauf in einem OSI TP-Auftragnehmer-Vorgang mit PEND ER, FR oder RS beendet wurde.

Beim Lesen einer Nachricht, die von einem Auftraggeber-Vorgang stammt, kann als Vorgangs-Status nur der Wert "O" auftreten.

Bei den Vorgangs-Stati D, E, I, R, T und Z wird keine Nachricht übertragen, d.h. die Rückgabelänge KCRLM hat den Wert 0.

Transaktionsstatus im Feld KCTAST/kcpta_state

Einträge bei Dialogen ohne verteilte Verarbeitung:

- "O" (open)
  Die Transaktion im lokalen Vorgang ist offen.
- "C" (closed)
  Entweder beim Vorgangsbeginn oder nach einem Sicherungspunkt.
- "R" (reset)
  Eine Rücksetznachricht wurde gelesen.

Einträge bei verteilter Verarbeitung nach Nachricht vom Partner-Vorgang:

- "C" (closed)
  Nur bei Kommunikation über LU6.1:
  Die Transaktion im Partner-Vorgang ist beendet. Diese Situation tritt dann ein, wenn im Partner-Vorgang ein PEND RE oder PEND FI erfolgte und der lokale Vorgang auf PEND RE steht.
- "H" (heuristic hazard)
  Nur bei der Kommunikation über das OSI TP-Protokoll:
  Weil die Kommunikation mit mindestens einem Kommunikationspartner unterbrochen wurde, besteht Unsicherheit über den Ausgang der Transaktion. Es kann nicht ausgeschlossen werden, dass einer der an der letzten Transaktion beteiligten Kommunikationspartner eine heuristische Entscheidung getroffen hat, die im Widerspruch zum tatsächlichen Ausgang der Transaktion steht.
- "I" (inactive)
  Es existiert keine Auftragnehmer-Transaktion, weil z.B. der Transaktionscode unbekannt ist oder keine Verbindung in der generierten Wartezeit belegt werden konnte.
- "M" (mismatch)
  Die Transaktion im entfernten Vorgang konnte nicht mit der Transaktion im lokalen Vorgang synchronisiert werden. Dies kann nach einem Timeout oder nach Beendigung und Start einer UTM-F-Anwendung auftreten. Bei der Kommunikation über das OSI TP-Protokoll kann diese Situation dann auftreten, wenn mindestens einer der an der Transaktion beteiligten Kommunikationspartner eine heuristische Entscheidung getroffen hat, die im Widerspruch zum tatsächlichen Ausgang der Transaktion steht.
- "O" (open)
  Die Transaktion im Partner-Vorgang ist offen.
- "P" (prepare to commit)
  Der Partner-Vorgang hat entweder selbst das Transaktionsende eingeleitet oder aber er fordert den lokalen Vorgang auf, das Transaktionsende einzuleiten.
- "R" (reset)
  Die Transaktion im Partner-Vorgang wurde zurückgesetzt.
- "U" (unknown)
  Nur möglich bei Kommunikation über OSI TP ohne globale Transaktionssicherung. Der Transaktionsstatus ist unbekannt.

Beim Lesen einer Nachricht, die von einem Auftraggeber-Vorgang stammt, können als Transaktionsstatus nur die Werte "C", "O", "P" oder "U" auftreten.

KDCS-Returncodes im Feld KCRCCC beim MGET-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde durchgeführt.
Die (Teil-)Nachricht wurde vollständig gelesen.

01Z Längenkonflikt: KCLA < KCRLM; der Nachrichtenbereich ist zu kurz, die (Teil-)Nachricht wurde abgeschnitten.

02Z Bei Nachrichten von einer Socket-USP-Anwendung oder einem HTTP-Client:
Längenkonflikt KCLA < KCRLM; der Nachrichtenbereich ist zu klein, der abgeschnittene Rest der Teilnachricht kann mit einem erneuten MGET-Aufruf gelesen werden.

03Z Bei Teilformaten:
KCMF/kcfn enthält nicht den Namen des nächsten zurückgesendeten Teilformats.

Bei verteilter Verarbeitung und Nachrichten von UPIC-Clients:
KCMF/kcfn enthält nicht das Formatkennzeichen bzw. den Namen der abstrakten Syntax der als Nächstes zu lesenden (Teil-)Nachricht.

Es wird keine (Teil-)Nachricht in den Nachrichtenbereich übertragen; die Felder KCRPI und KCRMF/kcrfn enthalten einen neuen Vorschlag für die nächste (Teil-)Nachricht.

05Z Bei Einzelformaten:
Am Bildschirm war ein anderes Format ausgegeben als in KCMF/kcfn eingetragen.
Die Nachricht wurde lt. Formatkennzeichen der letzten Ausgabe und nicht wie in KCMF/kcfn angegeben formatiert.

Im Zeilenmodus:

- Das erste Zeichen von KCMF/kcfn ist kein Leerzeichen

Nur auf BS2000-Systemen:

- Der Name des Editprofils ungültig.

10Z Die Nachricht wurde bereits vollständig gelesen

12Z (Nur im Auftraggeber-Vorgang möglich.)
Von der angegebenen Vorgangs-Id liegen keine (Teil-)Nachrichten (mehr) vor, aber von anderen Auftragnehmer-Vorgängen sind noch (Teil-)nachrichten vorhanden. Der Inhalt des Nachrichtenbereichs wurde nicht verändert. Die Felder KCRPI und KCRMF/kcrfn enthalten einen neuen Vorschlag, welche (Teil-)Nachricht als Nächstes gelesen werden könnte.

19Z Die Funktionstaste ist nicht generiert oder die zugeordnete Sonderfunktion ist ungültig.

20Z...
39Z

- Der Terminal-Benutzer hat eine Funktionstaste gedrückt, der bei der Generierung ein Returncode zugeordnet wurde,
- oder die Funktionstaste wurde von einem UPIC-Client ausgelöst.
- Wurde eine Funktionstaste von einem UPIC-Client ausgelöst oder eine Funktionstaste gedrückt, der eine Nachricht zugeordnet wurde, muss diese Nachricht mit einem weiteren MGET-Aufruf gelesen werden.

Nur auf BS2000-Systemen:

- KDCS*xx* (01 <= *xx* <= 20) eingegeben (Simulation einer Funktionstaste)

Weitere Returncodes sind dem DUMP zu entnehmen:

70Z Die Operation kann vom System nicht ausgeführt werden (System- bzw. Generierungsfehler), siehe KCRCDC.

71Z Im ersten Verarbeitungsschritt des ersten Teilprogrammlaufs eines Asynchron-Vorgangs wurde ein MGET-Aufruf gegeben, bzw. der INIT fehlt im Teilprogrammlauf.

72Z Die Angabe in KCOM ist ungültig.

73Z Die Längenangabe in KCLA ungültig.

77Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

78Z *Nur auf BS2000-Systemen:*
Der Event-Exit FORMAT meldet einen Fehler.

Eigenschaften des MGET-Aufrufs

- Verhalten bei Längenkonflikten

  Die tatsächliche Länge der (Teil-)Nachricht wird im Feld KCRLM zurückgegeben.

  Bei Längenkonflikten ist zu beachten: Bei KCRLM < KCLA werden nur KCRLM Zeichen (Bytes) in den Nachrichtenbereich übertragen. Der Inhalt des restlichen Nachrichtenbereichs ist undefiniert. In einem Teilprogramm kann nur eine Nachricht - eventuell bestehend aus mehreren Nachrichtenteilen - gelesen werden. Ist die Längenangabe in KCLA des Parameterbereichs kürzer, als die tatsächliche (Teil-)Nachricht, so geht der Rest (KCRLM-KCLA) verloren. Er kann mit einem nachfolgenden MGET nicht mehr gelesen werden. Ausnahme: Ist bei Nachrichten von einer Socket-USP-Anwendung oder einem HTTP-Client der Nachrichtenbereich zu klein (KCLA < KCRLM), kann der abgeschnittene Rest der Teilnachricht mit einem erneuten MGET-Aufruf gelesen werden.

  Beispiel

  Drei Teilnachrichten mit je 100 Byte sollen durch MGET-Aufrufe gelesen werden. Die Tabelle zeigt, wie sich verschiedene Angaben im Feld KCLA auswirken.

Benutzerangaben		**UTM-Rückgaben**			**Erläuterungen**
KCOP	**KCLA**	**KCRLM**	**Übertragene Länge im NB**	**KCRCCC**	
MGET	100	100	100	000	Die Teilnachricht wurde ordnungsgemäß empfangen.
MGET	50	100	50	01Z	Die Teilnachricht war länger als in KCLA angegeben, der Rest geht verloren.
MGET	150	100	100	000	Die Teilnachricht wurde ordnungsgemäß empfangen.
MGET	100	000	000	10Z	Eine vierte Teilnachricht war nicht vorhanden.

- Umwandlung von Kleinbuchstaben

  openUTM setzt beim MGET Kleinbuchstaben nicht automatisch in Großbuchstaben um. Eine Umsetzung lässt sich jedoch mit entsprechender Formatgenerierung erreichen.

  Auf BS2000-Systemen können Sie eine Umsetzung auch durch den Einsatz von Edit-Profilen erreichen.

- Nachrichten der Länge Null

  Nachrichten mit der Länge Null sind z.B. in folgenden Fällen möglich:

  - Zu Beginn eines Vorgangs wurde nur der Transaktionscode (ohne weitere Daten) gesendet.
  - In einem Folge-Teilprogramm soll eine Nachricht vom vorhergehenden Teilprogramm desselben Vorgangs gelesen werden und im vorhergehenden Teilprogramm wurde ein MPUT mit Länge 0 bzw. gar kein MPUT gegeben.
  - Ein Client-Programm oder ein Partner-Vorgang hat eine leere Nachricht gesendet.
  - Der Terminal-Benutzer hat eine Funktionstaste gedrückt, ohne eine Nachricht zuzuordnen,
  - Nur auf BS2000-Systemen: KDCS*xx* (01 <= *xx* <= 20) wurde eingegeben (Simulation einer Funktionstaste).
  - Nur auf BS2000-Systemen: Der Terminal-Benutzer hat eine leere Nachricht gesendet (DUE-Funktion mit leerem Bildschirm)

- Entfernen des Transaktionscodes

  Wurde zum Vorgangsstart ein Transaktionscode zusammen mit einer Nachricht angegeben und dabei keine Funktionstasten verwendet, so wird aus der Nachricht Folgendes entfernt:

  - bei unformatierter Eingabe der Transaktionscode inklusive nachfolgende Leerzeichen (MAX-Anweisung Parameter LEADING-SPACES)

  Nur auf BS2000-Systemen:

  - bei formatierter Eingabe mit *Formaten die ersten acht Zeichen der Nachricht (Transaktionscode)
  - bei formatierter Eingabe mit +Formaten die ersten zehn Zeichen (Attributfeld plus TAC),
  - bei formatierter Eingabe mit -Formaten die ersten acht Zeichen der Nachricht (Transaktionscode)
  - bei formatierter Eingabe mit Teilformaten die ersten acht (bei *Formaten) bzw. zehn (bei +Formaten) Zeichen der **ersten** Teilnachricht.

  Das Abschneiden des Transaktionscodes kann in einem INPUT-Exit verhindert werden.

  Bei einem MGET-Aufruf im Event-Service BADTACS wird der ungültige Transaktionscode **nicht** aus der Eingabenachricht entfernt, die gesamte Nachricht wird zur Verfügung gestellt. Dies gilt auch dann, wenn der ungültige Transaktionscode einer Funktionstaste zugeordnet ist.

- Empfangen von Teilformaten

  Jedes Teilformat muss mit einem eigenen MGET gelesen werden.
  openUTM liefert nach dem INIT in KCRMF/kcrfn den Namen des ersten Teilformats, in das Daten eingetragen wurden. Dieser Name ist beim MGET in KCMF/kcfn anzugeben. MGET liefert in KCRMF/kcrfn den Namen des nächsten Teilformats mit Eingabedaten, der beim folgenden MGET in KCMF/kcfn anzugeben ist. Beim letzten Teilformat mit Eingabedaten steht in KCRMF/kcrfn nochmals der in KCMF/kcfn angegebene Name, siehe Beispiel. Das letzte Teilformat erkennt man an gleichen Einträgen in KCMF/kcfn und KCRMF/kcrfn oder am Rückkehrcode 10Z beim folgenden MGET.

  Wurden in keines der Teilformate Daten eingegeben, so liefert bereits der erste MGET-Aufruf im Rückgabebereich KCRCCC=10Z, KCRLM=0, KCRMF=Leerzeichen.

  Wird in KCMF/kcfn ein anderer Name eingetragen als vorher in KCRMF/kcrfn geliefert, so

  - schreibt openUTM keine Daten in den Nachrichtenbereich,
  - setzt openUTM KCRLM=0,
  - setzt openUTM den Returncode 03Z in KCRCCC und
  - schreibt openUTM in KCRMF/kcrfn nochmals den ’richtigen’ Formatnamen.

  Nur auf BS2000-Systemen: Beachten Sie, dass die Art und Weise, wie Teilformate übertragen werden, auch von FHS-Startparametern abhängt, siehe FHS-Handbuch. Wird z.B. bei Teilformaten in kein Teilformat etwas eingegeben, so liefert openUTM bei bestimmten FHS-Startparametern beim ersten MGET KCRCCC = 10Z und in KCRMF/kcrfn Leerzeichen.

  Beispiel

  Die drei Teilformate *TEIL1, *TEIL2 und *TEIL3 sollen durch MGET-Aufrufe gelesen werden; beachten Sie die Rückgaben in KCRMF.

Benutzerangaben		**UTM-Rückgaben**		**Erläuterungen**
KCOP	**KCMF/kcfn**	**KCRMF/kcrfn**	**KCRCCC**	
INIT		*TEIL1	000	
MGET	*TEIL1	*TEIL2	000	1. Teilformat lesen
MGET	*TEIL2	*TEIL3	000	2. Teilformat lesen
MGET	*TEIL3	*TEIL3	000	3. Teilformat lesen
MGET	*TEIL3	*TEIL3	10Z	Nachricht war schon vollständig gelesen

  Das letzte Teilformat erkennt man an gleichen Einträgen in KCMF/kcfn und KCRMF/kcrfn oder am Rückkehrcode 10Z beim folgenden MGET.

Besonderheiten des MGET-Aufrufs bei verteilter Verarbeitung

- Bei der Kommunikation über das OSI TP-Protokoll wird das Formatkennzeichen zur Übergabe des Namens der abstrakten Syntax verwendet; Leerzeichen stehen dabei für die abstrakte Syntax von UDT. In allen Fällen, in denen das Anwendungsteilprogramm nicht mit UDT arbeiten möchte, muss die Umsetzung der Nachricht von der lokalen Darstellung in die Transfer Syntax bzw. umgekehrt - gemäß den Regeln der abstrakten Syntax - durch die Anwendung selbst durchgeführt werden. Diesen Vorgang bezeichnet man als Encodieren bzw. Decodieren der Nachricht. Dazu kann sich die Anwendung der Hilfe eines ASN1-Compilers bedienen.

  Das Codieren und Decodieren von Nachrichten im UDT-Format übernimmt openUTM.
- Bei der Kommunikation über das LU6.1-Protokoll überträgt openUTM zwar das Formatkennzeichen, formatiert die Nachricht aber nicht: Die Partner-Anwendungen tauschen immer nur Nettodaten aus. Beim MPUT kann im Feld KCMF/kcfn ein beliebiger Name angegeben werden. Dieser Name wird dem lesenden Teilprogramm nach dem INIT bzw. nach dem vorhergehenden MGET im Feld KCRMF/kcrfn angezeigt und muss beim MGET-Aufruf im Feld KCMF/kcfn angegeben werden.
- Die Returncodes für Funktionstasten (19Z bis 39Z) können beim MGET-Aufruf im Auftragnehmer-Vorgang nicht vorkommen, da der Auftraggeber-Vorgang keine entsprechenden Sonderfunktionen an den Auftragnehmer-Vorgang weiterleiten kann.
- Der MGET-Aufruf liefert im KDCS-Rückgabebereich den Vorgangs-Status und den Transaktionsstatus des Partner-Vorgangs.
- Wenn bei Kommunikation über LU6.1 die Bottom-Up-Strategie (siehe "Programmierregeln und Empfehlungen") nicht eingehalten wird, kann ein Vorgangs-Wiederanlauf mit dem Senden einer Nachricht vom Auftraggeber-Vorgang an den Auftragnehmer-Vorgang beginnen. Dann wird das Folge-Teilprogramm im Auftragnehmer-Vorgang gestartet. Dieses kann mit MGET die Nachricht vom letzten Sicherungspunkt lesen und erhält vom Auftraggeber den Vorgangs-Status "O" und den Transaktionsstatus "C". Es kann nach dem INIT-Aufruf am Vorgangskennzeichen KCKNZVG/kccv_status im KB-Kopf erkennen, dass es sich um einen Vorgangs-Wiederanlauf handelt.

Besonderheiten des MGET-Aufrufs bei Kommunikation mit einem UPIC-Client

Bei der Kommunikation überträgt openUTM zwar das Formatkennzeichen, formatiert die Nachricht aber nicht: Zwischen dem UPIC-Client und der Anwendung werden immer nur Nettodaten ausgetauscht. Beim Senden einer Nachricht kann als Formatkennzeichen ein beliebiger Name angegeben werden. Dieser Name wird dem lesenden Teilprogramm nach dem INIT bzw. nach dem vorhergehenden MGET im Feld KCRMF/kcrfn angezeigt und muss beim MGET-Aufruf im Feld KCMF/kcfn angegeben werden.

Besonderheiten des MGET-Aufrufs bei Kommunikation mit einer Socket-USP-Anwendung oder einem HTTP-Client

Eine Teilnachricht von einer Socket-USP-Anwendung oder einem HTTP-Client kann mit mehreren MGET-Aufrufen gelesen werden. Anhand des Returncodes kann erkannt werden, ob eine (Teil-)Nachricht vollständig gelesen wurde. Der Returncode 02Z zeigt an, dass eine Teilnachricht noch nicht vollständig gelesen wurde. Durch Vergleich von KCLA und KCRLM können Sie ermitteln, wie groß der Rest der Teilnachricht ist. Der Returncode 000 zeigt an, dass die

(Teil-)Nachricht vollständig gelesen wurde und der nächste MGET eine neue (Teil-)Nachricht lesen wird.

Eigenschaften einer Rücksetznachricht

Eine Rücksetznachricht stammt immer von einem Teilprogramm, das mit PEND RS beendet wurde. Sie wird nach Vorgangs-Wiederanlauf dem Teilprogramm zugestellt, das nach dem Sicherungspunkt als Folge-Teilprogramm gestartet wird. Die Rücksetznachricht muss mit dem ersten MGET gelesen werden. Aufgrund der Rücksetznachricht kann das Teilprogramm gezielt reagieren und dadurch ein nochmaliges Rücksetzen der Transaktion vermeiden. Das Teilprogramm erkennt einen Vorgangs-Wiederanlauf daran, dass das Vorgangskennzeichen den Wert "R" annimmt. Es steht nach dem INIT-Aufruf im Feld KCKNZVG/kccv_status zur Verfügung. Die Rücksetznachricht wird nach der Verarbeitung bei Verbindungsverlust und bei KDCOFF gelöscht.

Beim MPUT-Aufruf geben Sie im KDCS-Parameterbereich an, ob beim Vorgangs-Wiederanlauf ein Bildschirmwiederanlauf durchgeführt wird (KCDF = KCRESTRT) oder nicht (KCDF enthält binär null). Wenn kein Bildschirmwiederanlauf angefordert wird, startet openUTM nach dem Rücksetzen der Transaktion sofort den beim letzten Sicherungspunkt angegebenen Teilprogrammlauf. Mit MGET kann die Rücksetznachricht gelesen werden.

MGET-Aufruf zum Lesen einer Statusinformation vom Auftragnehmer

Statusinformationen sind Nachrichten der Länge 0 die von openUTM intern erzeugt werden. Sie dienen ausschließlich dazu, bei verteilter Verarbeitung in Fehlersituationen den Status des Auftragnehmer-Vorgangs anzuzeigen. Sie werden im Auftraggeber-Vorgang mit MGET-Aufrufen unformatiert gelesen (Leerzeichen in KCMF /kcfn).

Musste die verteilte Transaktion zurückgesetzt werden, so wird beim Vorgangs-Wiederanlauf möglichst das Teilprogramm erneut gestartet, für welches beim Ende der letzten verteilten Transaktion eine Nachricht vorlag, bzw. für welches die nächste Eingabe vom Terminal bestimmt ist. Wenn zuerst ein Teilprogramm im Auftraggeber-Vorgang gestartet wird, dann schickt openUTM - wenn notwendig - eine Statusinformation an dieses Programm. Dabei sind folgende Punkte zu beachten:

- Statusinformationen gibt es von den Auftragnehmern, die das Rücksetzen der verteilten Transaktion verursacht haben und dadurch beendet wurden oder werden.
- Ist die Eingabenachricht beim Vorgangs-Wiederanlauf für dieses Teilprogramm bestimmt, dann muss mit dem 1. MGET die Eingabenachricht und erst mit dem 2. MGET die Statusinformation gelesen werden. Wenn allerdings die Eingabenachricht vom Auftragnehmer-Vorgang stammt und wenn diese Nachricht vom Auftragnehmer nicht gesendet wird, so erhält man lediglich eine Statusinformation von dem Auftragnehmer-Vorgang.
- Wenn die Eingabenachricht beim Vorgangs-Wiederanlauf für den Auftragnehmer-Vorgang bestimmt war und wenn dieser Vorgang nach Ablauf einer generierten Wartezeit nicht gestartet werden kann (z.B. wegen Verbindungsverlust), dann startet openUTM statt dessen das Folge-Teilprogramm im Auftraggeber-Vorgang und schickt diesem eine Statusinformation, die mit dem 1. MGET gelesen werden muss.
- Wird nach einem Vorgangs-Wiederanlauf des Auftraggeber-Vorgangs wieder ein Auftragnehmer-Vorgang adressiert und tritt wieder ein Fehler auf, so kann der Auftraggeber-Vorgang mehrfach auf den gleichen Sicherungspunkt zurückgesetzt werden. Da die Statusinformationen vom vorhergehenden Rücksetzen erhalten bleiben, kann man eventuell mehrere Statusinformationen bekommen. In diesem Fall erhält man jeweils beim MGET die Vorgangs-Id eines nächsten Vorgangs, von dem eine Statusinformation vorliegt.
- Es können Statusinformationen von unterschiedlichen Auftragnehmer-Vorgängen vorliegen. Diese Statusinformationen müssen in der von openUTM vorgeschlagenen Reihenfolge (KCRPI) gelesen werden.

- Bei verteilter Verarbeitung über OSI TP erhält man "Ersatznachrichten" auch wenn kein Vorgangs-Wiederanlauf stattfindet.
  Ohne globale Transaktionssicherung wird die Transaktion im Auftraggeber-Vorgang bei einem Fehler mit dem Auftragnehmer-Vorgang (z.B. Timeout) **nicht** zurückgesetzt. Bei globaler Transaktionssicherung wird die Transaktion im Auftraggeber-Vorgang nur dann nicht zurückgesetzt, wenn keine verteilte Transaktion mit dem Auftragnehmer offen ist, z.B. bei Ablauf des Timers für die Association-Belegung.

KDCS-Sonderfunktionen (BS2000-Systeme)

Die KDCS-Schnittstelle bietet "KDCS-Sonderfunktionen" als eine besondere Form der Eingabe am Terminal. Der Terminal-Benutzer aktiviert sie durch Eingabe der Zeichenfolge

(KDCSxx) xx= 01,...,20

wenn UTM die Eingabe für einen Folgeteilprogrammlauf erwartet. Es sind also maximal 20 KDCS-Sonderfunktionen möglich. Die KDCS-Sonderfunktionen sind als Ersatzeingabe für Terminals gedacht, die nicht über geeignete Tasten verfügen.

7.16 MPUT Senden einer Dialog-Nachricht

Mit dem Aufruf MPUT (message PUT) können Sie

- eine Dialog (Teil-)Nachricht an einen Client senden,
- eine (Teil-)Nachricht an ein nachfolgendes Teilprogramm des gleichen Verarbeitungsschritts oder eines angeketteten Vorgangs senden,
- eine Rücksetznachricht senden für den Vorgangs-Wiederanlauf nach PEND RS,
- die letzte Bildschirmausgabe eines gekellerten Vorgangs an das Terminal schicken, oder
- bei verteilter Verarbeitung im Auftraggeber-Vorgang eine (Teil-)Nachricht an einen Auftragnehmer-Vorgang senden, oder
- bei verteilter Verarbeitung im Auftragnehmer-Vorgang eine (Teil-)Nachricht an den Auftraggeber-Vorgang senden.
- eine Verarbeitungsquittung von einem OSI TP-Partner anfordern.
- eine negative Verarbeitungsquittung oder eine Fehlermeldung an einem OSI TP-Partner senden.
- eine Fehlermeldung erzeugen, die im Falle eines von openUTM veranlassten abnormalen Vorgangsendes (System PEND ER) an einen UPIC-Client, eine Socket-USP-Anwendung oder einen HTTP-Client gesendet wird.

In einem Asynchron-Vorgang darf eine MPUT-Nachricht nur an einen Auftragnehmer-Vorgang oder an ein Folge-Teilprogramm gerichtet sein.

Der Aufruf darf nicht in einem MSGTAC-Programm verwendet werden.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCLM	KCRN	KCMF/kcfn	KCDF
BS2000-Systeme: Nachricht im Formatmodus	"MPUT"	"NT"/"NE"	Länge	Leerzeichen	Formatkennzeichen	Bildschirmfunktion
BS2000-Systeme: Nachricht im Zeilenmodus	"MPUT"	"NT"/"NE"	Länge	Leerzeichen	Leerzeichen /Editprofil	Bildschirmfunktion
Unix-, Linux- und Windows-Systeme: Nachricht im Zeilenmodus	"MPUT"	"NT"/"NE"	Länge	Leerzeichen	Leerzeichen	—
Nachricht an Teilprogramm	"MPUT"	"NT"/"NE"	Länge	TAC	—	—
letzte Bildschirmausgabe eines gekellerten Vorgangs	"MPUT"	"PM"	Länge	Leerzeichen	Formatkennzeichen /Leerzeichen	Bildschirmfunktion
Rücksetznachricht senden	"MPUT"	"RM"	Länge	Reset-Id	binär null	binär null / KCRESTRT
Nachricht an AN-Vorgang (LU6.1)	"MPUT"	"NT"/"NE"	Länge	Vorgangs-Id	Formatkennzeichen /Leerzeichen	binär null
Nachricht an AG-Vorgang (LU6.1)	"MPUT"	"NT"/"NE"	Länge	Leerzeichen	Formatkennzeichen /Leerzeichen	beliebiger Wert
Nachricht an AN-Vorgang (OSI TP)	"MPUT"	"NT"/"NE"	Länge	Vorgangs-Id	Leerzeichen / abstrakte Syntax	0
Nachricht an AG-Vorgang (OSI TP)	"MPUT"	"NT"/"NE"	Länge	Leerzeichen	Leerzeichen / abstrakte Syntax	0
Anforderung einer Verarbeitungsquittung	"MPUT"	"HM"	0	Vorgangs-Id /Leerzeichen	Leerzeichen	0
Fehlermeldung oder negative Quittung	"MPUT"	"EM"	0	Vorgangs-Id /Leerzeichen	Leerzeichen	0
Fehlermeldung für UPIC-Clients, Socket-USP-Anwendungen oder HTTP-Clients	"MPUT"	"ES"	Länge	Leerzeichen	Leerzeichen /Formatkennzeichen	0

NT: Teilnachricht
NE: letzte Teilnachricht bzw. Gesamtnachricht.

Bei KCOM = HM/EM/ES/PM/RM müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, aus dem openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"MPUT"
KCOM	"NT"/"NE"/"PM"/"RM"/"HM"/"EM"/"ES"
KCLM	Länge in Byte/0
KCRN	Leerzeichen/TAC/Reset-Id/Vorgangs-Id
KCMF/kcfn	Formatkennzeichen / Leerzeichen / Name der abstrakten Syntax / Editprofil (nur auf BS2000-Systemen)
KCDF	Bildschirmfunktion/binär null
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_MPUTNT / KDCS_MPUTNE	(nb,kclm,kcrn,kcfn,kcdf)
KDCS_MPUTPM	(nb,kclm,kcfn,kcdf)
KDCS_MPUTRM	(nb,kclm,kcfn)
KDCS_MPUTHM / KDCS_MPUTEM	(nb,kcrn)
KDCS_MPUTES	(nb,kclm,kcfn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den MPUT-Aufruf ein:

KCOP

im Feld KCOP den Operationscode "MPUT".

KCOM

im Feld KCOM

- NT für Teilnachricht,
- NE für Gesamtnachricht bzw. letzte Teilnachricht,
- PM für die letzte Bildschirmausgabe eines gekellerten Vorgangs oder die Anforderung eines Vorgangs-Wiederanlaufs im Anmelde-Vorgang,
- RM für eine Rücksetznachricht,
- HM für die Anforderung einer Verarbeitungsquittung von OSI TP-Partnern,
- EM für eine Fehlermeldung oder eine negative Verarbeitungsquittung an OSI TP-Partner.
- ES für das Erzeugen einer Fehlernachricht an einen UPIC-Client, eine Socket-USP-Anwendung oder einen HTTP-Client.

KCLM

im Feld KCLM die Länge der Nachricht im Nachrichtenbereich, die gesendet werden soll (Länge Null darf auch angegeben werden).

KCRN

im Feld KCRN abhängig vom Empfänger der Nachricht:

- den Transaktionscode eines Folgeprogramms, wenn dieser MPUT eine Nachricht an ein Folgeprogramm der gleichen Anwendung sendet (dies gilt auch für einen PEND PA/PR/FC/SP-Aufruf).
- Leerzeichen, wenn dieser MPUT eine Dialog-Nachricht an einen Client sendet.
- Die Rücksetz-Id (Reset-Id), wenn eine Rücksetznachricht für den Vorgangs-Wiederanlauf gesendet werden soll. Die Reset-Id muss mit "<" beginnen (siehe Kapitel „MGET Empfangen einer Dialog-Nachricht", Abschnitt „ Eigenschaften einer Rücksetznachricht").
- Leerzeichen, wenn dieser MPUT eine Fehlermeldung für einen UPIC-Client erzeugt.
- Leerzeichen bei einer Antwort an den Auftraggeber-Vorgang,
- die Vorgangs-Id eines Auftragnehmer-Vorgangs, wenn dieser MPUT-Aufruf an einen Auftragnehmer-Vorgang gerichtet ist.

KCMF/kcfn

im Feld KCMF/kcfn

- Leerzeichen bei einer Nachricht im Zeilenmodus;
- Leerzeichen bei KCOM = PM mit KCLM = 0.
- Leerzeichen oder Formatkennzeichen bei Nachrichten an einen UPIC-Client oder einen LU6.1-Partner.
- Leerzeichen oder Name der abstrakten Syntax
  Bei Nachrichten an OSI TP-Partner muss in dem Feld der Name der abstrakten Syntax der Nachricht angegeben werden. Leerzeichen stehen dabei für die abstrakte Syntax von UDT; in diesem Fall wird als Transfer Syntax BER verwendet, die Encodierung der Nachricht übernimmt openUTM.
  Wird hier ein Wert ungleich Leerzeichen angegeben, dann muss die Nachricht an openUTM in encodierter Form, d.h. in der zu dieser abstrakten Syntax passenden Transfer Syntax, übergeben werden.

Nur auf BS2000-Systemen:

- Editprofil bei einer Nachricht im Zeilenmodus
- ein Formatkennzeichen bei einer Nachricht im Formatmodus oder bei KCOM = PM mit KCLM > 0. Soll ein Bildschirm wiederhergestellt werden, so muss das angegebene Format Bestandteil des wiederherzustellenden Bildschirms sein.

In allen anderen Fällen irrelevant

KCDF

im Feld KCDF eine Bildschirmfunktion, falls der Empfänger ein Terminal ist.

In den folgenden Fällen ist das Feld mit binär null zu versorgen:

- KCMF/kcfn enthält den Namen eines #Formats.
- Die Nachricht ist für einen Auftragnehmer-Vorgang bestimmt.
- Die Nachricht ist für einen OSI TP-Auftraggeber-Vorgang bestimmt.
- MPUT PM mit KCLM = 0 wird verwendet.
- Nur auf BS2000-Systemen: KCMF/kcfn enthält den Namen ein Editprofils.

Beim Senden einer Rücksetznachricht (KCOM = RM) muss KCRESTRT oder binär null angegeben werden.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie ein die Nachricht, die Sie ausgeben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLM die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim MPUT-Aufruf

Im Programm sind auswertbar:

000 Die Funktion wurde ausgeführt.

41Z Zu viele MPUT-Aufrufe:

- weiterer MPUT NT/NE nach MPUT NE/HM
- weiterer MPUT nach bzw. vor einem MPUT PM
- mehr als ein MPUT RM
- MPUT RM in der ersten Transaktion eines Vorgangs oder nicht erlaubter Wechsel des Eintrags in KCRN (bei mehreren Teilnachrichten muss der TAC des Folgeprogramms immer der gleiche sein)
- MPUT HM wurde als erster Aufruf an einen OSI TP-Partner gegeben.
- nach einem CTRL-Aufruf wurde ein MPUT HM-Aufruf an den gleichen Partner gegeben
- nach einem CTRL AB-Aufruf wurde ein MPUT-Aufruf an den gleichen Partner gegeben
- nur in einem OSI TP-Auftragnehmer-Vorgang: es wurde ein MPUT an den Auftraggeber gegeben und KCSEND enthält N.
- es wurden bereits 254 MPUT NT-Aufrufe an einen HTTP-Client gegeben und die Ausgabenachricht soll anschließend von einem HTTP-Exit Programm aufbereitet werden

45Z Der in KCMF/kcfn angegebene Wert ist ungültig. Möglicher Grund: Die angegebene abstrakte Syntax ist für den OSI-LPAP-Partner nicht generiert.

Weitere Returncodes sind dem DUMP zu entnehmen:

70Z Das System kann die Operation nicht ausführen (System- bzw. Generierungsfehler), siehe KCRCDC.

71Z In einem MSGTAC-Teilprogramm wurde ein MPUT-Aufruf gegeben oder das Teilprogramm enthält keinen INIT.

72Z Eintrag in KCOM ist ungültig oder in einem Asynchron-Teilprogramm wurde ein MPUT PM gegeben.

73Z Die Längenangabe in KCLM ist ungültig.

74Z Der Wert in KCRN ist ungültig, weil

- in einem Dialog-Vorgang KCRN einen Wert enthält, der kein TAC eines Dialogteilprogramms oder keine gültige Vorgangs-Id ist.
- in einem Asynchron-Vorgang KCRN einen Wert enthält, der kein TAC eines Asynchron-Teilprogramms oder keine gültige Vorgangs-Id ist.
- der Benutzer nicht berechtigt ist, den Teilprogrammlauf zu verwenden.
- in einem Asynchron-Teilprogramm KCRN mit Leerzeichen belegt wurde.
- bei einem MPUT HM das Ziel in KCRN kein OSI TP-Partner ist.
- bei einem MPUT EM das Ziel in KCRN kein OSI TP-Partner ist.
- bei einem MPUT ES in KCRN keine Leerzeichen angegeben wurden.

75Z Die Angabe in KCMF/kcfn ist ungültig. Mögliche Gründe:

- Das Formatkennzeichen in KCMF/kcfn wechselt oder ist ungültig.

Nur auf BS2000-Systemen:

- Das Editprofil ist nicht generiert oder das Editprofil wechselt bei Nachrichtenteilen (MPUT NT).

77Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

89Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist für KCOM PM/RM/HM/EM/ES ungleich binär null.

Eigenschaften des MPUT-Aufrufs

- Der Nachrichtenbereich wird beim Ausführen des Aufrufs durch openUTM nicht verändert.
- Maximale Nachrichtenlänge für MPUT NT/NE

  Bei Nachrichten an Terminals, an Transportsystem-Anwendungen vom Typ APPLI oder an ein nachfolgendes Teilprogramm darf die Gesamtnachricht höchstens so groß sein wie der Wert, der im Operanden NB der Steueranweisung MAX generiert wurde.

  Ansonsten ist die Länge einer Teilnachricht auf 32767 Byte beschränkt und die Länge der Gesamtnachricht unbegrenzt.
- Bei Ausgaben im Formatbetrieb können Sie Bildschirmfunktionen verwenden (siehe "Bildschirmausgabefunktionen im Formatmodus (BS2000-Systeme)").

  Nur auf BS2000-Systemen: Wenn Editprofile verwendet werden, muss KCDF mit binär null versorgt werden, sonst reagiert openUTM mit 70Z.
- In einem Verarbeitungsschritt dürfen mehrere Nachrichten ausgegeben werden, wenn die Nachrichten an Auftragnehmer-Vorgänge gerichtet sind und die Transaktion am Ende des Verarbeitungsschritts offen bleibt. In allen anderen Fällen kann höchstens **eine** Nachricht ausgegeben werden.

- Eine Nachricht darf aus mehreren Nachrichtenteilen bestehen, z.B. mehrere MPUT NT gefolgt von einem MPUT NE mit gleichem KCRN. Nachrichten für Auftragnehmer-Vorgänge können Sie parallel aufbauen, d.h. die Vorgangs-Id in KCRN darf wechseln. Jeder andere Wechsel des Nachrichtenziels ist nicht erlaubt, denn dadurch wird ein Nachrichtenende für alle noch nicht abgeschlossenen Nachrichten bewirkt. Nach einem nicht erlaubten Wechsel des Nachrichtenziels ist kein weiterer MPUT erlaubt. Nach dem Abschluss einer Nachricht mit MPUT NE oder MPUT HM ist kein weiterer MPUT mit dem KCRN erlaubt.
- Bei PEND KP/RE/FI/ER/FR und PGWT KP/CM wird die ganze Nachricht dem Kommunikationspartner übermittelt (ggf. auch formatiert).
- Bei PEND PA/PR/FC/SP werden die Nachricht bzw. die Teilnachrichten dem Folgeprogramm übergeben, dessen TAC in KCRN angegeben wurde (beim PEND und beim MPUT-Aufruf). Das Folgeprogramm muss jede Teilnachricht mit einem eigenen MGET lesen.
- Am Ende eines Verarbeitungsschritts werden alle noch nicht abgeschlossenen Nachrichten implizit durch openUTM abgeschlossen.
- In einem Teilprogrammlauf, der mit PEND PA/PR, PS, SP oder FC endet, darf der MPUT-Aufruf fehlen. In einem mit PEND KP/RE oder PGWT KP beendeten Teilprogrammlauf muss mindestens ein MPUT gegeben worden sein, ebenso muss in einem Dialog-Vorgang vor PEND FI/ER oder FR mindestens ein MPUT gegeben worden sein; dies gilt jedoch nicht, wenn in einem OSI TP-Server-Vorgang das Feld KCSEND den Wert "N" enthält.
  Im Anmelde-Vorgang mit anschließendem Vorgangs-Wiederanlauf darf der MPUT fehlen, openUTM beendet dann den zum Wiederanlauf anstehenden Vorgang abnormal.
- In einem Asynchron-Vorgang darf vor PEND FI/ER oder FR kein MPUT gegeben worden sein.
- Leere Nachrichten, d.h. KCLM=0, sind erlaubt.

  Wird die leere Nachricht an ein Terminal im Format-Modus gesendet, führt sie zur Ausgabe eines leeren Formats bzw. Teilformats.
  Auf BS2000-Systemen müssen Abhängigkeiten von den FHS Startparametern berücksichtigt werden, siehe FHS Handbuch.

  Leere Nachrichten sind auch bei verteilter Verarbeitung erlaubt.
- Eine leere Nachricht an eine Transportsystem-Anwendung vom Typ APPLI wird nicht gesendet.
- Bei Teilnachrichten an Socket-USP-Anwendungen muss jede Teilnachricht mit einem eigenen MPUT NT verschickt werden. Für jeden MPUT NT/NE wird ein eigener Nachrichtenteil erzeugt.
  Bei MPUT-Aufrufen mit KCLM=0 werden keine Nachrichten gesendet, es sei denn, es werden automatisch erzeugte USP-Header (siehe "Ausgabe-Nachrichten von openUTM") verwendet. In diesem Fall wird der Header auch bei MPUT NE/NT mit der Länge Null gesendet.
  Ausnahme: Wenn das Teilprogramm nur einen einzigen MPUT NE/NT enthält und KCLM=0 ist, wird auch kein Header gesendet.
- Bei HTTP-Clients werden alle mit MPUT NT erzeugten Teilnachrichten im Message Body der HTTP Response gesendet.
- Ist der Empfänger kein Terminal, dann wird die Nachricht transparent übertragen, d.h. die Nachricht darf beliebige Bitmuster enthalten.
- Eine leere Nachricht an eine UPIC-Anwendung wird wegen Übermittlung des Senderechtes gesendet.

- Senden von Teilnachrichten

  Eine Nachricht darf aus mehreren Nachrichtenteilen bestehen, z.B. mehrere MPUT NT gefolgt von einem MPUT NE mit gleichem KCRN. Nachrichten für Auftragnehmer-Vorgänge können Sie parallel aufbauen, d.h. die Vorgangs-Id in KCRN darf wechseln. Jeder andere Wechsel des Nachrichtenziels ist nicht erlaubt, denn dadurch wird ein Nachrichtenende für alle noch nicht abgeschlossenen Nachrichten bewirkt. Wurde die letzte Teilnachricht nicht mit NE in KCOM gekennzeichnet, so wird die Nachricht bei einem PEND automatisch abgeschlossen.

  Einen Wechsel zwischen Zeilen- und Formatmodus oder einen Wechsel des Editprofils beantwortet openUTM mit 75Z und Vorgangs-Abbruch.

- Senden von Teilformaten

  Bei einem Terminal im Formatmodus kann ein Bildschirm aus mehreren Teilformaten aufgebaut sein. Dabei ist jedes Teilformat mit MPUT NT auszugeben, das letzte Teilformat kann mit MPUT NE ausgegeben werden.

- Bildschirmausgabefunktionen (KCDF) dürfen nur beim **ersten** MPUT NT angegeben werden. Bei weiteren MPUT-Aufrufen muss KCDF binär null enthalten, sonst beendet openUTM den Vorgang mit 70Z in KCRCCC und K606 in KCRCDC.

  Erlaubte Bildschirmausgabefunktionen bei Teilformaten:

KCREPL	Bildschirm löschen. Die Funktion ist anzugeben, wenn ein Bildschirm neu aufgebaut werden soll. Ist KCREPL nicht gesetzt, dann wird ein am Bildschirm vorhandenes Teilformat durch ein neues überschrieben. Falls dasselbe Teilformat erneut ausgegeben wird, werden nur die Feldinhalte ersetzt (wie bei KCERAS).
KCALARM	Akustisches Signal
KCREPR	Drucken auf Hardcopy-Drucker.
KCERAS	ungeschützte Felder löschen (näheres siehe Abschnitt „Einsatz von Teilformaten (BS2000-Systeme)“.

- Besonderheiten des MPUT RM-Aufrufs

  MPUT RM ist auch erlaubt, wenn vorher MPUT NT/NE oder MPUT PM-Aufrufe gegeben wurden. Die Länge der MPUT-RM-Nachricht ist auf den 32767 Byte beschränkt.
  In einem Teilprogrammlauf darf nur eine Rücksetznachricht ausgegeben werden. Andere MPUT-Aufrufe sind vor bzw. nach einem MPUT RM erlaubt. Rücksetz-Nachrichten werden beim Abmelden eines Benutzers gelöscht. Nach einem Vorgangs-Wiederanlauf enthält das Feld KCRPI Leerzeichen.

- Besonderheiten des MPUT PM-Aufrufs:

  Mit MPUT PM gibt openUTM die letzte Ausgabenachricht eines gekellerten Vorgangs auf dem Bildschirm aus. Dabei gilt:

  - Bei KCLM = 0 kommt die Ausgabe unverändert auf den Bildschirm, bei KCLM > 0 wird sie überschrieben (höchstens bis zur angegebenen Länge), aber in voller Länge gesendet. Die Länge der MPUT-PM-Nachricht ist durch den Wert beschränkt, der im Operanden NB der Steueranweisung MAX generiert wurde.
  - Bei Ausgabe-Nachrichten im Zeilenmodus muss immer KCLM = 0 (und KCMF/kcfn = Leerzeichen) angegeben werden.
  - Das Teilprogramm muss mit PEND FI beendet werden, sonst bricht openUTM den Vorgang mit 82Z ab.
  - Der Anmelde-Vorgang muss am Vorgangsende diese Variante des MPUT-Aufrufs benutzen, wenn ein Vorgangs-Wiederanlauf durchgeführt werden soll.

- Abstrakte Syntax bei verteilter Verarbeitung über das OSI TP-Protokoll

  Wird bei der Kommunikation mit einem Partner über das OSI TP-Protokoll in KCMF/kcfn ein Wert ungleich Leerzeichen angegeben, dann muss dieser Wert für den Partner als Name einer abstrakten Syntax generiert sein. In diesem Fall muss das Anwendungsteilprogramm die Nachricht an openUTM in encodierter Form übergeben, d.h. die Umsetzung in die der abstrakten Syntax zugeordnete Transfer Syntax muss von der Anwendung durchgeführt werden. Dazu kann sich die Anwendung der Hilfe eines ASN1-Compiler bedienen. Abstrakte Syntaxen mit den Namen "CCR" oder "OSITP" sind nicht zulässig.
- Bei der Kommunikation mit einem UPIC-Client oder über das LU6.1-Protokoll überträgt openUTM zwar das Formatkennzeichen, formatiert die Nachricht aber nicht: Die Partner-Anwendungen tauschen immer nur Nettodaten aus. Beim MPUT kann im Feld KCMF/kcfn ein beliebiger Name angegeben werden. Dieser Name wird dem lesenden Teilprogramm nach dem INIT bzw. nach dem vorhergehenden MGET im Feld KCRMF/kcrfn angezeigt und muss beim MGET-Aufruf im Feld KCMF/kcfn angegeben werden.
- MPUT-Aufruf im Auftraggeber-Vorgang
  - Mit einem MPUT-Aufruf kann ein Auftraggeber-Vorgang einen Auftragnehmer-Vorgang in einer Partner-Anwendung starten oder eine Nachricht an einen bereits gestarteten Auftragnehmer-Vorgang senden.
    Als Ziel ist im Feld KCRN die Vorgangs-Id anzugeben, die dem Auftragnehmer-Vorgang beim APRO DM-Aufruf zugeordnet wurde.
  - Innerhalb eines Teilprogrammlaufs des Auftraggeber-Vorgangs dürfen Nachrichten mit MPUT entweder nur an einen LTERM-Partner (KCRN=Leerzeichen), an das Folge-Teilprogramm (KCRN=TAC) oder an einen oder mehrere Auftragnehmer-Vorgänge (KCRN=Vorgangs-Id) gesendet werden.
  - Im Auftraggeber-Vorgang muss KCDF beim MPUT binär null enthalten.
  - Jede mit MPUT NT ausgegebene Teilnachricht an einen Auftragnehmer-Vorgang muss dort mit einem eigenen MGET gelesen werden.
  - In einem Teilprogrammlauf darf vor PEND PR bzw. PA keine (Teil-)Nachricht an einen Auftragnehmer-Vorgang gesendet werden, andernfalls bricht openUTM den Auftraggeber-Vorgang mit dem Returncode KCRCCC=82Z ab.
- MPUT-Aufruf im Auftragnehmer-Vorgang
  - Die Versorgung des KDCS-Parameterbereiches ist die gleiche wie beim MPUT-Aufruf für ein Terminal.
  - Als Bildschirmfunktion ist bei Kommunikation über LU6.1 im Feld KCDF ein beliebiger Wert zulässig, welcher dem Auftraggeber-Vorgang beim MGET übergeben wird. Bei Teilnachrichten wird nur der KCDF-Wert der ersten Teilnachricht übertragen.
    Bei Kommunikation über das OSI TP-Protokoll muss KCDF mit binär null versorgt werden.
  - Jede mit MPUT NT ausgegebene Teilnachricht an einen Partner-Vorgang muss dort mit einem eigenen MGET gelesen werden.
- Besonderheiten des MPUT HM-Aufrufs

  Mit MPUT HM kann ein Programmlauf eine Verarbeitungsquittung von einem OSI TP- Partner anfordern. Für die Verwendung des MPUT HM gelten folgende Regeln:

- Der Aufruf darf nur benutzt werden, wenn die Handshake-Funktion für die Kommunikation ausgewählt wurde, sonst bricht openUTM den Vorgang mit 72Z ab. Ein Auftragnehmer-Vorgang erhält beim INIT im KB-Kopf angezeigt, ob Handshake erlaubt ist.
- Einen MPUT HM an einen Partner, der nicht den Transaktionsstatus O oder U hat, lehnt openUTM mit 72Z ab.
- Vor einem MPUT HM muss mindestens ein MPUT NT an den Partner gegeben worden sein.
- Der MPUT HM bewirkt den Nachrichtenabschluss für den Kommunikationspartner.
- Dem Aufruf können keine Daten mitgegeben werden (KCLM = 0).
- Nach einem MPUT HM ist nur ein PEND KP oder ein PGWT KP erlaubt. Bei allen anderen PEND-Varianten reagiert openUTM mit dem Returncode 82Z.
- Nach einem MPUT HM darf kein CTRL PR/PE-Aufruf mehr an den gleichen Partner gerichtet werden.

- Besonderheiten des MPUT EM-Aufrufs
  - Mit dem Aufruf MPUT EM kann einem OSI TP-Partner ein Fehler gemeldet werden. Soll mit dem Aufruf eine Handshake-Anforderung negativ quittiert werden, muss der MPUT EM als erster MPUT an den Partner gegeben werden, der eine Verarbeitungsquittung verlangt. Der Aufruf muss in derselben Transaktion erfolgen, in der die Handshake-Anforderung gelesen wurde. Andernfalls wird eine positive Quittung gesendet.
  - Dem Aufruf können keine Daten mitgegeben werden (KCLM = 0).
  - Einen MPUT EM an einen Partner, der nicht den Transaktionsstatus O oder U hat, lehnt openUTM mit 72Z ab.
- Besonderheiten des MPUT ES Aufrufs
  - Mit dem Aufruf MPUT ES (error system) kann in einem Dialog-Teilprogramm eine Fehlermeldung für einen UPIC-Client, eine Socket-USP-Anwendung oder einen HTTP-Client erzeugt werden. Diese Fehlernachricht wird nur gesendet, wenn der Vorgang von openUTM abnormal beendet wird (system PEND ER).
  - Eine mit MPUT ES erzeugte Fehlernachricht bleibt, sofern sie nicht mit einem weiteren MPUT ES überschrieben wird, gültig, bis der Vorgang mit der Ausgabe einer Dialog-Nachricht an den UPIC-Client, die Socket-USP-Anwendung oder einen HTTP-Client beendet wird. Bei einer Vorgangs-Kettung (PEND FC) ist die Fehlernachricht also auch im geketteten Vorgang gültig.
  - Jeder weitere MPUT ES überschreibt die zuletzt erzeugte Fehlernachricht. Ein MPUT ES mit der Länge 0 löscht die Fehlernachricht.
  - Die Länge der MPUT ES Nachricht ist durch den Wert beschränkt, der im Operanden NB der Steueranweisung MAX generiert wurde.
  - Wird die Transaktion zurückgesetzt, wird die Fehlernachricht auf den Stand des letzten Sicherungspunktes zurückgesetzt.

7.17 PADM Administrieren von Druckausgaben und Druckern

Der Aufruf PADM (printer administration) dient zur Administration der zu einem Druckersteuer-LTERM gehörenden Drucker.

PADM bietet folgende Funktionen:

- Quittungsmodus für ein Druckersteuer-LTERM ein- oder ausschalten.
  Unix-, Linux- und Windows-Systeme
  In UTM-Cluster-Anwendungen wirkt diese Funktion Cluster-global.
- eine Druckausgabe bestätigen oder wiederholen
- die Zuordnung Drucker zu LTERM-Partner ändern.
  Unix-, Linux- und Windows-Systeme
  In UTM-Cluster-Anwendungen ist diese Funktion verboten.
- den Druckerstatus ändern, d.h. Drucker sperren und wieder freigeben, Verbindung zu einem Drucker ab- oder aufbauen.
  Unix-, Linux- und Windows-Systeme
  In UTM-Cluster-Anwendungen wirkt das Sperren und Freigeben Cluster-global.
- Informationen über einen Drucker in den Nachrichtenbereich lesen
- Informationen über zu quittierende Druckausgaben lesen

> Mit openUTM wird ein Musterprogramm ausgeliefert, das Sie hier einsetzen können. Näheres finden Sie im openUTM-Handbuch „Anwendungen administrieren“ im Abschnitt „Teilprogramm KDCPADM“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich						
	KCOP	KCOM	KCLA	KCRN	KCLT	KCACT	KCADRLT
Druckausgabe bestätigen	"PADM"	"OK"	0	Control-Id	LTERM-Name	binär null	binär null
Druckausgabe wiederholen	"PADM"	"PR"	0	Control-Id	LTERM-Name	binär null	binär null
Quittiermodus ändern	"PADM"	"AT"/"AC"	0	Control-Id /Leerzeichen	LTERM-Name	binär null	binär null
Druckerzuordnung ändern	"PADM"	"CA" *Unix-, Linux- und Windows-Systeme* In UTM-Cluster-Anwendungen nicht erlaubt	0	Control-Id	LTERM-Name	binär null	LTERM-Name
Druckerzustand ändern	"PADM"	"CS"	0	Control-Id	LTERM-Name	ON/OFF /CON /DIS	binär null
Informieren über Druckausgabe	"PADM"	"AI"	44	Control-Id /Leerzeichen	LTERM-Name	binär null	binär null
Informieren über Drucker	"PADM"	"PI"	34	Control-Id /Leerzeichen	LTERM-Name	binär null	binär null

Alle nicht verwendeten Felder des KDCS-Parameterbereichs müssen mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier müssen Sie die Adresse des Nachrichtenbereichs bereitstellen, in den openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"PADM"
KCOM	"OK"/"PR"/"AT"/"AC"/"CA"/"CS"/"AI"/"PI"
KCLA	Länge in Byte/0
KCRN	Control-Id/Leerzeichen
KCLT	Name des Druckersteuer-LTERM
KCACT	"ON"/"OFF"/"CON"/"DIS"/binär null
KCADRLT	binär null/LTERM-Name

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_PADMOK/KDCS_PADMPR/KDCS_PADMAT/KDCS_PADMAC	(nb,kcrn,kclt)
KDCS_PADMCA	(nb,kcrn,kclt,kcadrlt)
KDCS_PADMCS	(nb,kcrn,kclt,kcact)
KDCS_PADMAI/KDCS_PADMPI	(nb,kcla,kcrn,kclt)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten/-
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Länge
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Folge-Control-Id/Leerzeichen

In den KDCS-Parameterbereich tragen Sie für den PADM-Aufruf ein:

KCOP

im Feld KCOP den Operationscode PADM.

KCOM

im Feld KCOM

- OK: Bestätigen einer Druckausgabe
- PR: Wiederholen einer Druckausgabe
- AC: Quittungsmodus einschalten.
  Unix-, Linux- und Windows-Systeme
  AC wirkt in UTM-Cluster-Anwendungen Cluster-global.
- AT: Quittungsmodus ausschalten, d.h. in den Automatikmodus zurückschalten.
  Unix-, Linux- und Windows-Systeme
  AT wirkt in UTM-Cluster-Anwendungen Cluster-global.
- CA: Zuordnung zu einem anderen LTERM.
  Unix-, Linux- und Windows-Systeme
  CA ist in UTM-Cluster-Anwendungen verboten.
- CS: Zustand eines Druckers ändern.
  Unix-, Linux- und Windows-Systeme
  CS zum Sperren und Freigeben wirkt in UTM-Cluster-Anwendungen Cluster-global.
- AI: Informationen über zu bestätigende Druckausgaben lesen
- PI: Informationen über einen Drucker lesen

KCLA

im Feld KCLA die Länge, in der die Daten in den Nachrichtenbereich übertragen werden sollen:

- bei KCOM=AI: 44
- bei KCOM=PI: 34

bei allen anderen Varianten wird 0 eingetragen.

KCRN

im Feld KCRN

- bei KCOM = OK/PR/CA/CS die Control-Id eines Druckers, wie sie in der PTERM-Anweisung bei KDCDEF generiert wurde,
- bei KCOM = AT/AC/AI/PI die Control-Id eines Druckers (gemäß PTERM-Anweisung) oder Leerzeichen, wenn die Operation für alle Drucker gelten soll, die dem Druckersteuer-LTERM zugeordnet sind.

KCLT

im Feld KCLT den Namen des Druckersteuer-LTERMs. Falls dies nicht das eigene Terminal ist, dann muss die eigene Benutzerkennung administrationsberechtigt sein.

KCACT

im Feld KCACT bei KCOM = CS die Aktion, die durchgeführt werden soll:

- ON: Drucker wird entsperrt (STATUS = ON);
  Unix-, Linux- und Windows-Systeme
  wirkt in UTM-Cluster-Anwendungen Cluster-global.
- OFF: Drucker wird gesperrt (STATUS = OFF);
  Unix-, Linux- und Windows-Systeme
  wirkt in UTM-Cluster-Anwendungen Cluster-global.
- CON: Logische Verbindung zum Drucker aufbauen
- DIS: Logische Verbindung zum Drucker abbauen

Bei allen anderen Varianten wird binär null eingetragen

KCADRLT

im Feld KCADRLT bei KCOM = CA den Namen des LTERM-Partners, der dem Drucker neu zugeordnet werden soll. Der Drucker wird durch seine Control-Id identifiziert, die in KCRN angegeben wird.
Bei allen anderen Varianten wird binär null eingetragen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

2. Parameter

   Die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs müssen Sie auch angeben, wenn Sie in KCLA 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

bei KCOM = AI/PI im angegebenen Nachrichtenbereich die Nachricht in der tatsächlichen, höchstens aber in der gewünschten Länge.

KCRLM

im Feld KCRLM die tatsächliche Länge der Nachricht, ggf. in Abweichung von der angeforderten Länge in KCLA des Parameterbereichs.

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

im Feld KCRMF/kcrfn bei KCOM = AI/PI die Control-Id des nächsten Druckers, der dem Druckersteuer-LTERM zugeordnet ist oder Leerzeichen, wenn es der letzte zugeordnete Drucker ist.

KDCS-Returncodes im Feld KCRCCC beim PADM-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde ausgeführt (bei KCOM = AI/PI) bzw. der Administrationsauftrag wurde angenommen (bei KCOM = OK/PR/CA/CS/AT/AC).

01Z Längenkonflikt: KCLA < KCRLM, die Nachricht wurde abgeschnitten.

40Z Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, keine Berechtigung für diesen Aufruf) siehe KCRCDC.

Unix-, Linux- und Windows-Systemen:
Operation ist in einer UTM-Cluster-Anwendung nicht erlaubt.

42Z Der Eintrag in KCOM ist ungültig.

43Z Die Längenangabe in KCLA ist negativ bzw. ungültig.

44Z Die in KCRN angegebene Control-Id ist ungültig oder im Bereich des Steuerterminals gibt es keinen Drucker mit dieser Control-Id.

46Z Die Angabe in KCLT ist ungültig, da unter diesem Namen kein Druckersteuer-LTERM definiert ist.

47Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

55Z Die Angabe in KCACT ist ungültig.

56Z Die Angabe in KCADRLT ist ungültig.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Rückgaben im Nachrichtenbereich bei PADM AI

Für die Strukturierung des Nachrichtenbereichs beim Aufruf PADM AI steht eine Sprachspezifische Datenstruktur zur Verfügung, für COBOL im COPY-Element KCPADC und für C/C++ in der Include-Datei *kcpad.h*. Sie hat folgenden Aufbau:

Byte	Feldname COBOL/C/C++	Bedeutung	
1 - 8	KCACKCID	Control-Id des Druckers	
9 - 16	KCGENUID	UTM-Benutzerkennung des Auftragserzeugers	
17 - 24	KCDPUTID	Auftrags-Id (von openUTM vergeben)	
25 - 33	KCGENTIM[1]	Zeitpunkt des FPUT-/DPUT-Aufrufs in der Form *dddhhmmss*:	
25 - 27	KCGENDOY	*ddd*	Tag im Jahr (Wertebereich 001 - 366)
28 - 29	KCGENHR	*hh*	Stunde (Wertebereich 00 - 23)
30 - 31	KCGENMIN	*mm*	Minute (Wertebereich 00 - 59)
32 - 33	KCGENSEC	*ss*	Sekunde (Wertebereich 00 - 59)
34 - 42	KCSTTIM[1]	bei zeitverzögertem Auftrag die gewünschte Startzeit in der Form *dddhhmmss*:	
34 - 36	KCSTDOY	*ddd*	Tag im Jahr (Wertebereich 001 - 366)
37 - 38	KCSTHR	*hh*	Stunde (Wertebereich 00 - 23)
39 - 40	KCSTMIN	*mm*	Minute (Wertebereich 00 - 59)
41 - 42	KCSTSEC	*ss*	Sekunde (Wertebereich 00 - 59)
		Bei einem Auftrag ohne Zeitverzögerung werden Leerzeichen eingetragen.	
43	KCPOSMSG	Y	Positiver Quittungsauftrag vorhanden
		N	Kein positiver Quittungsauftrag vorhanden
44	KCNEGMSG	Y	Negativer Quittungsauftrag vorhanden
		N	Kein negativer Quittungsauftrag vorhanden

[1] Für C/C++ sind die zusammenfassenden Felder KCGENTIM und KCSTTIM nicht definiert, wohl aber die spezifischen Felder für Tag/Stunde/Minute/Sekunde.

Wenn keine Druckausgabe zu bestätigen ist, dann schreibt openUTM Leerzeichen in den Nachrichtenbereich.

Rückgaben im Nachrichtenbereich bei PADM PI

Für die Strukturierung des Nachrichtenbereichs beim Aufruf PADM PI steht eine Sprachspezifische Datenstruktur zur Verfügung, für COBOL im COPY-Element KCPADC und für C/C++ in der Include-Datei *kcpad.h*. Sie hat folgenden Aufbau:

Byte	Feldname COBOL/C /C++	Bedeutung
1 - 8	KCPRTCID	Control-Id des Druckers
9 - 11	KCSTATE	Status des Druckers: ON oder OFF
12	KCCON	Druckerverbindung: Y = Drucker verbunden N = Drucker nicht verbunden
13 - 14	KCPRTMOD	Druckmodus: AT = Automatik-Modus AC = Quittungsmodus
15 - 22	KCLTRMNM	LTERM-Name des zugehörigen Druckers
23 - 28	KCFPMSGS	Anzahl der Ausgabeaufträge für diesen Drucker (ohne die Anzahl in KCDPMSGS).
29 - 34	KCDPMSGS	Anzahl der für diesen Drucker anstehenden zeitgesteuerten Aufträge, deren jeweilige Zielzeit noch nicht erreicht worden ist.

Eigenschaften des PADM-Aufrufs

- Die PADM-Aufrufe mit der Modifikation AT/AC werden erst beim Transaktionsende ausgeführt. Bei den Aufrufen PADM OK/PR/CS/CA wird erst beim Transaktionsende überprüft, ob der Aufruf überhaupt ausführbar ist. Daher garantiert KCRCCC = 000 nach einem solchen Aufruf noch nicht, dass der Aufruf erfolgreich ausgeführt werden kann, denn in der Zwischenzeit könnte z.B. die Verbindung zum Drucker verlorengehen. Ob ein PADM OK/PR /CS/CA-Aufruf erfolgreich war, kann man durch einen PADM AI- bzw. PADM PI-Aufruf in einer nachfolgenden Transaktion überprüfen.
- Ist bei einem PADM OK/PR-Aufruf keine Druckausgabe zu bestätigen, dann liefert PADM OK oder PADM PR den Returncode 40Z.
- Die Druckerzuordnung darf mit PADM CA nur dann geändert werden, wenn die beiden beteiligten LTERMs zur Aufrufzeit nicht mit der Anwendung verbunden sind, andernfalls antwortet openUTM mit 40Z.
- Bei gesperrtem Drucker lehnt openUTM einen PADM CS-Aufruf zwecks Verbindungsaufbau (KCACT = CON) mit 40Z ab.

7.18 PEND Beenden eines Teilprogramms

Mit dem Aufruf PEND (program end) beenden Sie ein Teilprogramm. Jedes UTM-Teilprogramm, einschließlich der Event-Services (BADTACS, MSGTAC, SIGNON) muss über den PEND-Aufruf verlassen werden (Ausnahme: Event-Exits). Je nach Art des Programms wird eine der Varianten des PEND-Aufrufs verwendet.

Bei verteilter Verarbeitung müssen die PEND-Aufrufe von Auftraggeber-Vorgang und Auftragnehmer-Vorgang aufeinander abgestimmt werden, siehe Kapitel „Programmaufbau bei verteilter Verarbeitung“.

Die Abkürzungen in Feld KCOM sind aus folgenden Begriffen entstanden:

PS	**p**rogram and **s**ign(on)
PA/**PR**	**pr**ogr**a**m
KP	**k**ee**p**
RE	**re**turn
SP	**s**ynchronization **p**oint
FI	**fi**nish
FC	**f**inish and **c**ontinue
RS	**r**e**s**et
ER	**er**ror
FR	**f**inish and **r**eset

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich		
	KCOP	KCOM	KCRN
Anmelde-Vorgang nach Berechtigungsprüfung im Folge-Teilprogramm fortsetzen	"PEND"	"PS"	TAC des Folge-Teilprogramms
Verarbeitungsschritt im Folge-Teilprogramm fortsetzen	"PEND"	"PA"/"PR"	TAC des Folge-Teilprogramms
Verarbeitungsschritt beenden ohne Transaktionsende	"PEND"	"KP"	TAC des Folge-Teilprogramms
Verarbeitungsschritt beenden mit Transaktionsende	"PEND"	"RE"	TAC des Folge-Teilprogramms
Transaktion beenden; Verarbeitungsschritt fortsetzen	"PEND"	"SP"	TAC des Folge-Teilprogramms
Verarbeitungsschritt, Transaktion und Vorgang beenden	"PEND"	"FI"	—
Transaktion + Vorgang beenden; Verarbeitungsschritt im angeketteten Vorgang fortsetzen	"PEND"	"FC"	TAC des Folge-Teilprogramms
Transaktion zurücksetzen	"PEND"	"RS"	Leerzeichen
Vorgang abbrechen, die Transaktion zurücksetzen, Speicherabzug (Dump) anfordern und Anwendungsprogramm neu starten	"PEND"	"ER"	—
Vorgang abbrechen und die Transaktion zurücksetzen, kein Speicherabzug, kein Neustart des Anwendungsprogramms	"PEND"	"FR"	Leerzeichen

ℹ Bei KCOM = PS/FC/SP/RS müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"PEND"
KCOM	"PS"/"PA"/"PR"/"KP"/"RE"/"FI"/"FC"/"RS"/"ER"/"FR"/"SP"
KCRN	Folge-Transaktionscode/Leerzeichen/ —

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_PENDER/KDCS_PENDFI/KDCS_PENDFR/KDCS_PENDRS	()
KDCS_PENDFC/KDCS_PENDKP/KDCS_PENDPA/KDCS_PENDPR/KDCS_PENDPS /KDCS_PENDRE/KDCS_PENDSP	(kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich tragen Sie für den PEND-Aufruf ein:

KCOP

im Feld **v** den Operationscode PEND.

KCOM

im Feld KCOM die Variante des PEND-Aufrufs:

- PS: Fortsetzung des Anmelde-Vorgangs in einem Folgeprogramm
- PR / PA: Fortsetzung des Verarbeitungsschritts in einem Folgeprogramm
- KP: Ende des Verarbeitungsschritts ohne Transaktionsende
- RE: Ende des Verarbeitungsschritts mit Transaktionsende
- SP: Ende der Transaktion, Fortsetzung des Verarbeitungsschritts in einem Folgeprogramm
- FI: Vorgangsende
- FC: Vorgangsende mit Vorgangs-Kettung

- RS: Transaktion wird zurückgesetzt (mit anschließendem Vorgangs-Wiederanlauf, wenn ein Aufsetzpunkt existiert).
- ER: Vorgangsende wegen Fehler. Der Vorgang wird abnormal beendet, die Transaktion zurückgesetzt und ein DUMP erzeugt.
- FR: Vorgangsende wegen Fehler. Der Vorgang wird abnormal beendet und die Transaktion zurückgesetzt (ohne DUMP!).

KCRN

im Feld KCRN je nach Variante

- bei PEND KP/RE:
  den TAC des Folge-Teilprogramms in dem nach dem Empfang der nächsten Eingabenachricht die Verarbeitung fortgesetzt werden soll.
- bei PEND PA/PR/PS/SP:
  den TAC des Folge-Teilprogramms in dem die Verarbeitung des gleichen Verarbeitungsschritts fortgesetzt werden soll.
- bei PEND RS/FR:
  Leerzeichen
- bei PEND ER/FI:
  irrelevant.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe unten.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim PEND-Aufruf

Diese Returncodes sind nur dem DUMP zu entnehmen:

000	Die Operation wurde ausgeführt (bei angefordertem PEND ER).
70Z	Die Operation PEND konnte nicht ausgeführt werden (System- bzw. Generierungsfehler, Deadlock, Timeout), siehe KCRCDC.
71Z	In diesem Teilprogrammlauf wurde noch kein INIT gegeben.

72Z KCOM enthält eine ungültige Operationsmodifikation bei Aufruf

- im MSGTAC-Programm,
- im Anmelde-Vorgang,
- außerhalb des Anmelde-Vorgangs,
- im Server-Vorgang,
- im Asynchron-Vorgang

oder die Operationsmodifikation steht im Widerspruch

- zur Datenbanktransaktion,
- zum verwendeten Kommunikationsprotokoll,
- zum Status des Anmelde-Vorgangs
- zum Warten auf eine DGET-Nachricht

74Z das Feld KCRN enthält einen Wert, der kein TAC oder kein Folge-TAC ist,

- der Benutzer ist nicht berechtigt, den TAC zu verwenden,
- ein Wechsel zwischen Dialog- und Asynchron-TAC liegt vor,
- ein Wechsel zwischen Teilprogrammen mit KDCS- und X/OPEN- API liegt vor,
- bei einem PEND RS,FR ist KCRN nicht mit SPACES belegt
- beim Warten auf eine DGET-Nachricht ist der in KCRN angegebene TAC in keiner TAC-Klasse

81Z Die Angabe in KCRN beim PEND PA/PR/FC/SP/PS (TAC des Folgeprogramms) steht im Widerspruch zur Angabe in KCRN beim MPUT (z.B. MPUT TAC1 und PEND xx TAC2).

82Z Die Angabe in KCOM oder KCRN des PEND widerspricht den Angaben beim vorausgehenden MPUT:

- MPUT mit KCRN=Leerzeichen bei PEND PA/PR/PS/SP/FC
- MPUT mit KCRN=TAC bei PEND KP/RE/FI/ER/FR
- MPUT mit KCRN=RESET-ID und kein PEND RS
- MPUT mit KCRN=Vorgangs-Id bei PEND PA/PR/FI/ER/FR.
- MPUT mit KCRN=Leerzeichen an einen HTTP-Client und kein PEND FI

83Z MPUT Aufruf fehlt:

- Vor einem PEND KP/RE/FI/ER/FR hat das Dialog-Programm keinen MPUT gegeben
- vor einem PEND RS in einer Folgetransaktion wurde keine Rücksetznachricht mit MPUT RM gesendet
- vor einem mit PEND FI beendeten Teilprogramm in einem Anmelde-Vorgang mit Vorgangs-Wiederanlauf wurde kein MPUT PM gegeben.

86Z Fehlender KDCS-Aufruf beim Message Queuing:

- Auftrags-Komplex beim PEND-Aufruf nicht abgeschlossen
- zu einem DPUT NI-Aufruf war kein Auftrag erzeugt worden
- an einen mit APRO AM adressierten Vorgang war kein Asynchron-Auftrag gesendet worden.

87Z Der PEND-Aufruf steht im Widerspruch zum Transaktions- oder Vorgangs-Status eines entfernten Vorgangs.

89Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null (nur bei KCOM = PS/FC/RS/SP).

Die folgende Tabelle zeigt die PEND-Varianten und die damit verknüpften Aktionen.

Variante	Bedeutung	Aktionen von openUTM
PS	Anmelde-Vorgang soll im Folgeprogramm fortgesetzt werden. Kein Sicherungspunkt!	• Berechtigungsdaten prüfen • eventuell Zwischendialog zur Abfrage von Passwort, Ausweisinformation führen. • MPUT dieses Teilprogramms für das Folgeteilprogramm bereitstellen bzw. auf Pagepool sichern, wenn ein Zwischendialog notwendig ist. • TAC des Folgeteilprogramms aus dem Feld KCRN übernehmen und Folgeprogramm sofort oder nach Ende des Zwischendialogs starten.
PA oder PR	Verarbeitungsschritt soll im Folgeprogramm fortgesetzt werden. Die Transaktion bleibt offen. Kein Sicherungspunkt!	• MPUT dieses Teilprogramms für das Folgeprogramm bereitstellen. • TAC des Folgeprogramms aus dem Feld KCRN übernehmen. Bei TAC-Klassensteuerung wird das Folge-Teilprogramm sofort gestartet, wenn es zu derselben TAC-Klasse gehört, ansonsten wird es verzögert gestartet. Soll auf eine DGET-Nachricht gewartet werden, startet openUTM das Folgeteilprogramm erst dann, wenn für diese Queue eine Nachricht eintrifft oder erneut in die Queue gestellt wird (Redelivery) oder wenn die maximale Wartezeit abgelaufen ist oder wenn die Queue gelöscht wurde (siehe DGET-Aufruf).

Variante	Bedeutung	Aktionen von openUTM
SP	Ende der Transaktion. Verarbeitungsschritt soll aber im nächsten Teilprogramm fortgesetzt werden. Sicherungspunkt!	• DB-Transaktion schließen • DPUT, FPUT, LPUT, PTDA, SPUT und SREL der Transaktion ausführen. • In der ersten Transaktion eines Asynchron-Vorgangs FGET-Nachricht löschen. • Mit DGET verarbeitete Nachrichten aus ihrer Queue löschen. • MPUT dieses Teilprogramms ausführen • TAC des Folgeprogramms aus dem Feld KCRN übernehmen. Bei TAC-Klassensteuerung wird das Folge-Teilprogramm sofort gestartet, wenn es zu derselben TAC-Klasse gehört, ansonsten wird es verzögert gestartet.
KP	Ende des Verarbeitungsschritts, ohne die Transaktion zu beenden. Sie soll im nächsten Verarbeitungsschritt fortgesetzt werden. Kein Sicherungspunkt!	• MPUT dieses Teilprogramms ausführen (ggf. Nachricht formatieren). • ggf. Daten im KB übernehmen und dem Folgeprogramm übergeben, sobald es initialisiert wird. • TAC des Folgeprogramms aus dem Feld KCRN übernehmen und Folgeprogramm starten, sobald die nächste Nachricht kommt.
RE	Ende des Verarbeitungsschritts und gleichzeitiges Ende der Transaktion. Der Vorgang soll in einem Folgeprogramm fortgesetzt werden. Sicherungspunkt!	• DB-Transaktion schließen • DPUT, FPUT, LPUT, PTDA, SPUT und SREL der Transaktion ausführen. • In der ersten Transaktion eines Asynchron-Vorgangs FGET-Nachricht löschen. • Mit DGET verarbeitete Nachrichten aus ihrer Queue löschen. • MPUT dieses Teilprogramms ausführen (ggf. Nachricht formatieren). • Betriebsmittel freigeben • TAC des Folgeteilprogramms aus dem Feld KCRN übernehmen und Folgeteilprogramm starten, sobald die nächste Nachricht kommt. • ggf. Daten im KB-Programmbereich übernehmen und dem Folgeteilprogramm übergeben, sobald es initialisiert wird.

Variante	Bedeutung	Aktionen von openUTM
FI	Ende eines Vorgangs und Ende der Transaktion. Sicherungspunkt!	• DB-Transaktion schließen • DPUT, FPUT, LPUT, PTDA, SPUT und SREL (für GSSB) der Transaktion ausführen • In der ersten Transaktion eines Asynchron-Vorgangs FGET-Nachricht löschen. • Mit DGET verarbeitete Nachrichten aus ihrer Queue löschen. • LSSBs des Vorgangs freigeben. • MPUT dieses Teilprogramms ausführen (ggf. Nachricht formatieren) • Betriebsmittel freigeben
FC	Ende eines Vorgangs und Ende der Transaktion, Verarbeitungsschritt soll im angeketteten Vorgang fortgesetzt werden. Sicherungspunkt!	• DB-Transaktion schließen • DPUT, FPUT, LPUT, PTDA, SPUT und SREL (für GSSB) der Transaktion ausführen • Mit DGET verarbeitete Nachrichten aus ihrer Queue löschen. • LSSBs des Vorgangs freigeben. • MPUT dieses Teilprogramms an das erste Teilprogramm des angeketteten Vorgangs übergeben • Betriebsmittel freigeben

Variante	Bedeutung	Aktionen von openUTM
RS	Rücksetzen einer Transaktion auf den letzten Sicherungspunkt	In der ersten Transaktion eines Vorgangs: • UTM- und DB-Transaktion zurücksetzen • Vorgang beenden (gegebenenfalls Meldung K034 an den Client) • nur bei Terminals und TS-Anwendungen: letzte Ausgabenachricht des vorherigen Vorgangs ausgeben (wenn vorhanden). • Asynchron-Vorgänge und Folge-Vorgänge (bei Vorgangs-Kettung) nach dem Rücksetzen erneut starten in einer Folgetransaktion eines Vorgangs: • UTM- und DB-Transaktion auf den letzten Sicherungspunkt zurücksetzen und ggf. erfolgt Bildschirmwiederanlauf mit Meldung K034. • das am letzten Sicherungspunkt adressierte Teilprogramm starten • Rücksetznachricht an dieses Teilprogramm übergeben für alle Transaktionen eines Vorgangs: • mit DGET verarbeitete Nachrichten können nochmal verarbeitet werden, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen noch nicht erreicht ist. Ansonsten werden sie gelöscht oder bei Nachricht einer TAC-Queue gegebenenfalls in die Dead Letter Queue gesichert. • Sind für den aktuellen TAC PGWT-Aufrufe erlaubt und der PEND RS wurde nicht vom Teilprogramm aufgerufen: Anwendungsprogramm neu starten

Variante	Bedeutung	Aktionen von openUTM
ER (ERror)	Ende des Vorgangs mit Speicherabzug und Neustart des Anwendungsprogramms. Rücksetzen auf letzten Sicherungspunkt (Ausnahme: MPUT)	• UTM-Dump schreiben • UTM- und DB-Transaktion zurücksetzen • Mit DGET verarbeitete Nachrichten können nochmal verarbeitet werden, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen noch nicht erreicht ist. Ansonsten werden sie gelöscht oder bei Nachricht einer TAC-Queue gegebenenfalls in die Dead Letter Queue gesichert. • In der ersten Transaktion eines Asynchron-Vorgangs: Die FGET-Nachricht kann nochmal verarbeitet werden, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen noch nicht erreicht ist. Ansonsten wird sie gelöscht oder gegebenenfalls in die Dead Letter Queue gesichert • Speicherabzug des Anwendungsprogramms veranlassen. • MPUT dieses Teilprogramms ausführen (ggf. Nachricht formatieren) • Anwendungsprogramm neu starten • Betriebsmittel freigeben
FR	Ende des Vorgangs, kein Speicherabzug, das Anwendungsprogramm bleibt geladen. Rücksetzen auf letzten Sicherungspunkt (Ausnahme: MPUT)	• UTM- und DB-Transaktion zurücksetzen • Mit DGET verarbeitete Nachrichten können nochmal verarbeitet werden, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen noch nicht erreicht ist. Ansonsten werden sie gelöscht oder bei Nachricht einer TAC-Queue gegebenenfalls in die Dead Letter Queue gesichert. • In der ersten Transaktion eines Asynchron-Vorgangs: Die FGET-Nachricht kann nochmal verarbeitet werden, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen noch nicht erreicht ist. Ansonsten wird sie gelöscht oder gegebenenfalls in die Dead Letter Queue gesichert. • MPUT dieses Teilprogramms ausführen (ggf. Nachricht formatieren) • Betriebsmittel freigeben

Eigenschaften des PEND-Aufrufs

- Nach einem PEND wird nicht in das aufrufende Programm zurückverzweigt. Daher kann dort der KDCS-Rückgabecode nicht ausgewertet werden.
- Im Fehlerfall, wenn der PEND nicht ordnungsgemäß auszuführen war, ruft openUTM intern PEND ER auf.
  - Bei Dialog-Verarbeitung wird eine Meldung an den Client gesendet. Die Transaktion wird zurückgesetzt und der Vorgang beendet. Man kann einen neuen Vorgang starten.
  - Ein Asynchron-Vorgang wird abgebrochen und eine Meldung auf die System-Protokolldatei SYSLOG geschrieben.
- Führte ein anderer Aufruf zu einem KDCS-Rückgabecode >= 70Z, so ruft openUTM intern PEND ER auf.
- Ruft ein Teilprogramm in einem Dialog-Vorgang PEND ER/FR auf, muss zuvor mit MPUT eine Nachricht an den Client bzw. an den Auftraggeber-Vorgang geschickt werden - mit einer Ausnahme: Für einen OSI TP-Auftragnehmer-Vorgang bei dem KCSEND den Wert "N" enthält, gilt das nicht. Wird ein PEND ER vom openUTM intern aufgerufen (KCRCCC >= 70Z), wird eine Standardnachricht ausgegeben. Wurde jedoch bei der Kommunikation mit einem UPIC-Client mit MPUT ES eine eigene Fehlernachricht erzeugt, wird stattdessen diese Fehlernachricht an den UPIC-Client gesendet.
- Vor einem PEND RS-Aufruf in einer Folgetransaktion muss mit MPUT RM eine Rücksetznachricht gesendet werden. Ein PEND RS sollte daher nicht in der ersten Transaktion eines Vorgangs abgesetzt werden, da dann keine Rücksetznachricht gelesen werden kann. Im Gegensatz zum PEND ER wird beim PEND RS kein DUMP geschrieben.
- Wird der PEND RS-Aufruf bei Kommunikation mit einem UPIC-Client eingesetzt, muss Folgendes beachtet werden:

  Falls die vorhergehende Transaktion mit PEND SP oder PEND FC beendet wurde, wird bei einem PEND RS die lokale Transaktion zurückgesetzt und der Vorgang mit dem beim PEND SP oder PEND FC spezifizierten Folge-Teilprogramm fortgesetzt (lokaler Vorgangs-Wiederanlauf). Falls die vorhergehende Transaktion **nicht** mit PEND SP/FC beendet wurde und der Vorgang unter einer Benutzerkennung **mit** Wiederanlauf-Eigenschaft läuft, so wird der Vorgang auf den letzten Sicherungspunkt zurückgesetzt und der Dialog mit dem UPIC-Client beendet. Unter der Benutzerkennung kann ein neuer OSI TP Dialog gestartet und ein Vorgangs-Wiederanlauf angefordert werden. In allen anderen Fällen beendet openUTM den Vorgang mit PEND FR.
- Wird der PEND RS-Aufruf bei verteilter Verarbeitung über das OSI TP-Protokoll ohne COMMIT-Funktionalität eingesetzt, muss Folgendes beachtet werden:
  - bei Aufruf in einem Auftragnehmer-Vorgang: Falls die vorhergehende Transaktion mit PEND SP beendet wurde, wird bei einem PEND RS die lokale Transaktion zurückgesetzt und der Vorgang mit dem beim PEND SP spezifizierten Folge-Teilprogramm fortgesetzt (lokaler Vorgangs-Wiederanlauf). Falls die vorhergehende Transaktion **nicht** mit PEND SP beendet wurde und der Vorgang unter einer Benutzerkennung **mit** Wiederanlaufeigenschaft läuft, so wird der Vorgang auf den letzten Sicherungspunkt zurückgesetzt und der Dialog mit dem Auftraggeber beendet. Unter der Benutzerkennung kann ein neuer OSI TP Dialog gestartet und ein Vorgangs-Wiederanlauf angefordert werden.In allen anderen Fällen beendet openUTM den Vorgang mit PEND FR.
  - bei Aufruf in einem Auftraggeber-Vorgang werden alle Auftragnehmer-Vorgänge dieses Vorgangs mit PEND FR beendet.
- Wird ein PEND RS-Aufruf in einer Transaktion gegeben, die zuvor mit PGWT CM beendet wurde, dann wird der Vorgang mit PEND FR beendet.

- Wurden Nachrichten oder Nachrichtenteile, die openUTM nach dem INIT für das Programm bereithielt, im Programm mit MGET bzw. FGET nicht gelesen, gehen sie beim PEND verloren (auch beim PEND PA bzw. PR). Das Gleiche gilt bei unvollständigem Lesen einer DGET-Nachricht: die restlichen, nicht gelesenen Nachrichtenteile der Nachricht gehen verloren.
- Wurde vor einem PEND RE/SP/FI/FC eine DB-Transaktion abgeschlossen, wird die Ausführung bis zum PEND RE/SP/FI/FC verzögert.
- Ein PEND KP sperrt Betriebsmittel (LSSBs, GSSBs, TLS, ULS, ggf. Bereiche in Datenbanken) über einen Verarbeitungsschritt hinaus.

  Deshalb wird empfohlen (falls nicht mit verteilter Verarbeitung gearbeitet wird)

  - den PEND KP sparsam zu verwenden, um die Belegungszeiten für globale Betriebsmittel kurz zu halten,
  - bei Verwendung des PEND KP die globalen Betriebsmittel erst im letzten Verarbeitungsschritt zu belegen.
- Wenn die Transaktion zurückgesetzt wird, geht eine durch einen PEND KP auszuführende MPUT-Ausgabe verloren. Der Vorgang wird auf den letzten Sicherungspunkt zurückgesetzt. Es wird sofort ein Vorgangswiederanlauf durchgeführt, sofern der Benutzer noch angemeldet ist.

  Wenn der Benutzer abgemeldet wurde, z.B. weil die Verbindung zum Client verlorenging, wird beim Anmelden ein Vorgangswiederanlauf durchgeführt, sofern die Benutzerkennung (in Anwendungen ohne Benutzerkennungen der LTERM-Partner)

  mit Wiederanlauf-Eigenschaft generiert ist (Generierungs-Operanden RESTART=YES der LTERM bzw. USER-Anweisung, siehe openUTM-Handbuch „Anwendungen generieren“). Nach einem Anwendungsende ist in stand-alone UTM-Anwendungen ein Vorgangswiederanlauf nur in UTM-S-Anwendungen möglich.
- Ein PEND SP ist bei verteilter Verarbeitung über LU6.1 nur erlaubt, wenn keine Partner-Vorgänge mit offenen Transaktionen vorhanden sind.
- Wird eine Nachricht an einen HTTP-Client gesendet, so muss der Vorgang mit PEND FI beendet werden, da ein Vorgang für einen HTTP-Client nur einen Dialog-Schritt umfassen darf.
- Ein PEND PA/PR oder SP kann in einem Dialog- oder Asynchron-Vorgang zu einem Prozesswechsel führen, wenn der Folgeteilprogrammlauf in einer anderen TAC-Klasse liegt als der den PEND aufrufende Teilprogrammlauf (siehe openUTM-Handbuch „Anwendungen generieren“, TAC-Klassen), oder wenn die Anwendung mit der Anweisung TAC-PRIORITIES generiert wurde.
- Ein PEND PS-Aufruf darf nur im Anmelde-Vorgang und nur, wenn es der Status des Anmelde-Vorgangs erlaubt, angegeben werden.
- Wird im Anmelde-Vorgang für einen UPIC-Client nach Empfang einer Nachricht vom Client der Aufruf PEND PA /PR oder PS ohne vorhergehenden MPUT-Aufruf ausgeführt, so kann das Folgeteilprogramm nicht gelesene Nachrichten(-teile) vom UPIC-Client noch lesen.
- Ein PEND FC beendet den UTM-Vorgang, aber nicht die UPIC-Conversation.Wird im Anmelde-Vorgang für einen UPIC-Client nach Empfang einer Nachricht vom Client der Aufruf PEND FC ohne vorhergehenden MPUT-Aufruf ausgeführt, so kann das Folgeteilprogramm noch nicht gelesene Nachrichten (-teile) vom UPIC-Client noch lesen. In diesem Fall erhält das erste Teilprogramm des geketteten Vorgangs als Vorgangskennzeichen im KBKOPF den Wert F (First), nicht C (Chained), da es eine Nachricht vom Client erhält.
- Wird ein Asynchron-Vorgang oder ein mit PEND FC geketteter Folge-Vorgang in seiner ersten Transaktion unterbrochen, dann startet openUTM den Vorgang erneut.
- Teilprogrammläufe in Asynchron-Vorgängen dürfen PEND KP und RE nur verwenden, wenn sie zuvor mit einem MPUT eine Nachricht für einen Auftragnehmer-Vorgang bereitgestellt haben.

- Bedenken Sie, dass beim PEND ER/FR im Auftragnehmer-Vorgang keine Nachricht an den Auftraggeber-Vorgang geschickt werden kann (wie beim PEND ER/FR an das Terminal). Trotzdem ist ein MPUT-Aufruf vor dem PEND ER/FR notwendig, sonst beendet openUTM den Vorgang. Der Auftraggeber-Vorgang erhält dann eine Statusinformation mit Vorgangs-Status Z (statt E).
- Bei verteilter Verarbeitung muss beim PEND-Aufruf der Vorgangs- und der Transaktions-Status des Partners beachtet werden. Weitere Informationen hierzu finden Sie im Kapitel „Programmaufbau bei verteilter Verarbeitung “.
- Soll auf eine DGET-Nachricht gewartet werden, muss ein PEND PA/PR oder PGWT PR folgen.

7.19 PGWT Wartepunkt im Programm setzen ohne Teilprogrammende

Mit dem Aufruf PGWT (program wait) wird das Teilprogramm auf einen internen Wartepunkt gesetzt, **ohne** es zu beenden. Dabei haben Sie die Möglichkeit,

- den Verarbeitungsschritt zu beenden ohne die Transaktion zu beenden
- weder den Verarbeitungsschritt noch die Transaktion zu beenden, um auf eine Nachricht an einer Service-gesteuerten Message Queue zu warten (kein vorheriger MPUT erlaubt!)
- die Transaktion zu beenden. Falls zuvor eine MPUT-Nachricht gesendet wird, wird der Verarbeitungsschritt beendet, ansonsten wird er fortgesetzt.
- die Transaktion zurückzusetzen und den Verarbeitungsschritt fortzusetzen

Das Teilprogramm wird immer vom selben Prozess fortgesetzt (Prozess-Id bleibt erhalten). Der PGWT-Aufruf ist vergleichbar mit einem PEND-Aufruf mit anschließendem impliziten INIT-Aufruf. Geben Sie beim Aufruf im KDCS-Parameterbereich den Parameter KCLI mit einem Wert größer als 0 an, dann liefert openUTM Information über Anwendung, System, Kommunikationspartner. Der PGWT-Aufruf ist dann vergleichbar mit einem PEND-Aufruf mit anschließendem impliziten INIT PU-Aufruf.

Die folgende Tabelle zeigt die Bedeutung des PGWT-Aufrufs und die damit verknüpften Aktionen:

Variante	Bedeutung	Aktionen von openUTM
PGWT KP	Ende des Verarbeitungsschritts, ohne Teilprogramm- oder Transaktionsende, ggf. Informationen anfordern (KCLI > 0)	• MPUT-Nachricht senden • Programm fortsetzen sobald die Antwort vorliegt • bei KCLI > 0: angeforderte Informationen im Nachrichtenbereich bereitstellen
PGWT PR	Warten ohne Beenden von Verarbeitungsschritt, Teilprogramm oder Transaktion, ggf. Informationen anfordern (KCLI > 0)	• Programm fortsetzen sobald eine Nachricht an der Queue vorliegt • bei KCLI > 0: angeforderte Informationen im Nachrichtenbereich bereitstellen
PGWT CM	Transaktion beenden ohne Beenden des Teilprogramms, ggf. Informationen anfordern (KCLI > 0)	• Wenn MPUT gegeben wurde: MPUT-Nachricht senden und Programm fortsetzen, sobald die Antwort vorliegt. • Ohne MPUT: Verarbeitung ohne Warten fortsetzen, Ausnahme bei OSI TP siehe "PGWT-Aufruf in einem verteilten OSI TP-Vorgang mit Commit". • bei KCLI > 0: angeforderte Informationen im Nachrichtenbereich bereitstellen
PGWT RB	Transaktion zurücksetzen, ggf. Informationen anfordern (KCLI > 0)	• Rücksetzen einer Transaktion • Verarbeitung fortsetzen, Ausnahme bei OSI TP siehe " PGWT-Aufruf in einem verteilten OSI TP-Vorgang mit Commit ". • bei KCLI > 0: angeforderte Informationen im Nachrichtenbereich bereitstellen

Bei allen PGWT-Aufrufen bleibt der lokale Prozess während der Wartezeit bzw. der Kommunikation belegt. Bei den Aufrufen PGWT KP/PR bleiben darüber hinaus Betriebsmittel gesperrt. Verwenden Sie PGWT-Aufrufe daher sparsam und wohlüberlegt.

Versorgung des KDCS-Parameterbereichs (1. Parameter)

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich		
	KCOP	KCOM	KCLI
Beenden des Verarbeitungsschritts ohne Teilprogramm- oder Transaktionsende	"PGWT"	"KP"	0 / Länge des Nachrichtenbereichs
Warten ohne Beenden von Verarbeitungsschritt und Transaktion	"PGWT"	"PR"	0 / Länge des Nachrichtenbereichs
Beenden der Transaktion ohne Teilprogrammende	"PGWT"	"CM"	0 / Länge des Nachrichtenbereichs
Rücksetzen der Transaktion und Fortsetzen des Verarbeitungsschritts	"PGWT"	"RB"	0 / Länge des Nachrichtenbereichs

Versorgung des 2. Parameters (nur notwendig bei KCLI > 0)

Hier geben Sie die Adresse des Nachrichtenbereichs an, in den openUTM die angeforderte Information schreiben soll. Für die Strukturierung des Nachrichtenbereichs können Sie die gleiche Datenstruktur verwenden wie für den INIT PU-Aufruf, also für COBOL das COPY-Element KCINIC, für C/C++ die Include-Datei *kcini.h.*

Im Header der Datenstruktur geben Sie die Versionsnummer der Struktur an und wählen aus, welche Information openUTM zurückliefern soll.

Der genaue Aufbau der Datenstruktur ist beim INIT PU-Aufruf in Kapitel "INIT Initialisieren eines Teilprogramms" beschrieben.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"PGWT"
KCOM	"KP"/"PR"/"CM"/"RB"
KCLI	0 / Länge in Byte

Versorgen des Headers des Nachrichtenbereichs (nur notwendig bei bei KCLI > 0)	
Feldname im Nachrichtenbereich	Inhalt
KCVER/if_ver	Versionsnummer (7)
KCDATE/dattim_info	Anfordern von Datum und Uhrzeit (Y/N)
KCAPPL/appl_info	Anfordern von Anwendungsinformation (Y/N)
KCLOCALE/locale_info	Anfordern von Locale Information (Y/N)
KCOSITP/ositp_info	Anfordern von OSI TP Informationen (Y/N)
KCENCR/encr_info	Anfordern von Verschlüsselungs-Informationen (Y/N)
KCMISC/misc_info	Anfordern von verschiedenen Informationen (Y/N)
KCHTTP/http_info	Anfordern von HTTP Informationen (Y/N)

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich (nur notwendig bei KCLI > 0)

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_PGWTKP	()
KDCS_PGWTKP_PU/KDCS_PGWTPR/KDCS_PGWTCM/KDCS_PGWTRB	(nb,kcli)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten (nur bei KCLI > 0)
Feldname im KB-Rückgabebereich	
KCRLM	Länge der übertragenen Daten (nur bei KCLI > 0)
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Formatkennzeichen / Leerzeichen
KCRPI	Vorgangs-Id / Reset-Id / Leerzeichen

Im KDCS-Parameterbereich machen Sie für den PGWT-Aufruf folgende Einträge:

KCOP

im Feld KCOP den Operationsnamen "PGWT".

KCOM

im Feld KCOM die Variante des PGWT-Aufrufs:

- KP: Ende des Verarbeitungsschritts ohne Transaktionsende.
- PR: Warten ohne Beenden von Verarbeitungsschritt und Transaktion
- CM: Ende der Transaktion
- RB: Rücksetzen der Transaktion

KCLI

im Feld KCLI die Länge des Nachrichtenbereichs, in den openUTM die Information übertragen soll. Die Länge müssen Sie als Anzahl Bytes angeben. openUTM überträgt maximal so viel Bytes Information in den Nachrichtenbereich wie in KCLI angegeben wird.

Ist KCLI größer null, dann muss bei dem Aufruf als 2. Parameter die Adresse des Nachrichtenbereichs angegeben werden. Die Information entspricht der des INIT PU- Aufrufs.

Alle nicht verwendeten Felder des Parameterbereichs müssen binär null enthalten.

Versorgen des Headers des Nachrichtenbereichs (nur notwendig bei KCLI > 0):

KCVER/if_ver

Im Feld KCVER/if_ver geben Sie die Versionsnummer der Datenstruktur an, die derzeit aktuelle Version ist 7.

KCDATE/dattim_info

Das Feld KCDATE/dattim_info versorgen Sie mit Y, falls Sie Informationen über Datum und Uhrzeit des Starts der Anwendung und des Teilprogrammlaufs haben wollen, sonst geben Sie N an.

KCAPPL/appl_info

Das Feld **v**rsorgen Sie mit Y, wenn Sie Informationen über Anwendung, System und Kommunikationspartner anfordern, sonst geben Sie N an.

KCLOCALE/locale_info

Das Feldvversorgen Sie mit Y, wenn Sie Informationen über die Sprachumgebung des LTERM-Partners anfordern, sonst geben Sie N an.

KCOSITP/ositp_info

Das Feld KCOSITP/ositp_info versorgen Sie mit Y, wenn Sie OSI TP-spezifische Informationen benötigen, sonst geben Sie N an.

KCENCR/encr_info

Das Feld **v**orgen Sie mit Y, wenn Sie Informationen über Verschlüsselungsverfahren benötigen, die zwischen dem Client und der UTM-Anwendung gefahren werden, sonst geben Sie N an. (Der Verschlüsselungsmechanismus kann vereinbart werden. Siehe openUTM-Handbuch „Anwendungen generieren“.)

KCMISC/misc_info

Das Feld KCMISC/misc_info versorgen Sie mit Y, wenn Sie verschiedene Informationen (z.B. Anzahl der Asynchron-Nachrichten in der Queue des Benutzers, Passwortgültigkeit, Zeitpunkt der letzten Anmeldung) benötigen, sonst geben Sie N an.

KCHTTP/http_info

Das Feld **KCHTTP/http_info** versorgen Sie mit Y, wenn Sie HTTP-spezifische Informationen benötigen, sonst geben Sie N an.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   als 1. Parameter: die Adresse des KDCS-Parameterbereichs.

2. Parameter

   als 2. Parameter (nur notwendig bei KCLI ungleich 0):
   die Adresse des Nachrichtenbereichs, in den openUTM die Informationen schreiben soll. Für die Rückgaben steht Ihnen die Datenstruktur KCINIC für COBOL und die Include-Datei *kcini.h* für C/C++zur Verfügung (Beschreibung siehe "INIT Initialisieren eines Teilprogramms").

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

nur bei KCLI > 0:
im angegebenen Nachrichtenbereich die Information in der tatsächlichen, höchstens aber in der in KCLI angegebenen Länge. Die vom Aufruf PGWT gelieferten Daten entsprechen den Rückgabeinformationen des Aufrufs INIT PU (Beschreibung siehe "INIT Initialisieren eines Teilprogramms").

KCRLM

nur bei KCLI > 0:
im Feld KCRLM die tatsächliche Länge der von openUTM übergebenen Information, sofern KCRCCC = 000.
Bei KDCRCCC = 07Z enthält KCRLM die Länge der insgesamt zur Verfügung stehenden Daten. Bei KCRCCC > 40Z ist KCRLM = 0.

KCRCCC

im Feld KCRCCC den KDCS-Returncode (Länge 3 Byte).
Die möglichen Returncodes und ihre Bedeutung siehe unten.

KCRCDC

im Feld **KCRCDC** den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

nur bei PGWT CM mit vorherigem MPUT und PGWT KP (ähnlich wie beim INIT-Aufruf):

- bei einer Nachricht von einem Terminal:

  Leerzeichen (im Zeilenmodus) oder den Formatnamen (im Formatmodus) der letzten Bildschirmausgabe, also den Namen der beim MPUT des letzten Dialog-Schritts im Feld KCMF/kcfn angegeben wurde. Bestand die letzte Ausgabe aus mehreren Teilformaten, so enthält KCRMF/kcrfn den Namen des ersten Teilformats, in das Daten eingegeben wurden. Wurden in keines der Teilformate Daten eingegeben, so enthält KCRMF/kcrfn den Namen des ersten Teilformats.

  Nur auf BS2000-Systemen: Wurde bei der letzten Bildschirmausgabe ein Editprofil verwendet, so enthält KCRMF/kcrfn dieses Editprofil.
- bei einer Nachricht von einer TS-Anwendung: Leerzeichen
- bei einer Nachricht vom LU6.1-Partner oder UPIC-Client:
  das Formatkennzeichen des ersten Nachrichtenteils, das der LU6.1-Partner oder UPIC-Client beim Senden angegeben hat.

  Besonderheit im LU6.1-Auftraggeber-Vorgang:
  Leerzeichen, wenn für die in KCRPI angegebene Vorgangs-Id eine
  Statusinformation vorliegt
- bei verteilter Verarbeitung über OSI TP:

  Wurde der Teilprogrammlauf auf Grund eines verteilten Dialogs gestartet, so enthält KCRMF/kcrfn im Auftraggeber-Vorgang den Namen der abstrakten Syntax, die der Nachricht vom Auftraggeber zugeordnet wurde; enthält das Feld Leerzeichen, dann ist als abstrakte Syntax die UDT-Syntax ausgewählt.

  Im Auftragnehmer-Vorgang enthält KCRMF/kcrfn den Namen der abstrakten Syntax, die der Nachricht von dem in KCRPI beschriebenen AN-Vorgang zugeordnet wurde; enthält das Feld Leerzeichen, dann ist als abstrakte Syntax die UDT-Syntax ausgewählt, oder von dem Partner liegt eine Fehlernachricht vor.

KCRPI

nur bei PGWT CM mit vorherigem MPUT und PGWT KP (ähnlich wie beim INIT-Aufruf):

- Bei einer Nachricht von einem LTERM-Partner: Leerzeichen.
- Im Auftraggeber-Vorgang bei verteilter Verarbeitung:
  die Vorgangs-Id des Auftragnehmer-Vorgangs, wenn eine Nachricht vom Auftragnehmer vorliegt.

- im Auftragnehmer-Vorgang bei verteilter Verarbeitung:
  Leerzeichen.

KDCS-Returncodes im Feld KCRCCC beim PGWT-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde erfolgreich ausgeführt. Falls beim Aufruf ein Nachrichtenbereich angegeben war (KCLI > 0), wurde die angeforderte Information in der vollen Länge in den Nachrichtenbereich übertragen.
07Z	Die Funktion wurde ausgeführt, der bereitgestellte Nachrichtenbereich ist zu kurz (Länge in KCLI zu kurz). Es wurde keine oder nur unvollständige Information zurückgegeben.
40Z	Bei PGWT CM: die Funktion wurde nicht ausgeführt. Es ist ein Fehler aufgetreten, der ein Vorsetzen der aktuellen Transaktion nicht erlaubt. Die Transaktion wurde implizit mit PGWT RB zurückgesetzt.
48Z	Nur bei KCLI größer Null: Ungültige Version der Datenstruktur.

Weitere Returncodes sind dem DUMP zu entnehmen:

70Z	Die Operation PGWT konnte nicht ausgeführt werden (System- bzw. Generierungsfehler, Deadlock, Timeout).
71Z	In diesem Programm wurde noch kein INIT gegeben oder der Aufruf wurde von einem MSGTAC-Programm ausgeführt.
72Z	Der Eintrag in KCOM ist ungültig oder es wurde KCOM = CM/RB angegeben und in der aktuellen verteilten Transaktion ist ein Partner beteiligt, mit dem über das LU6.1-Protokoll kommuniziert wird.
77Z	Die beim Aufruf angegebene Adresse des Nachrichtenbereichs ist ungültig.
82Z	Vor einem PGWT KP wurde ein MPUT an einen Teilprogrammlauf gegeben.
83Z	Vor einem PGWT KP hat das Teilprogramm keinen MPUT gegeben oder vor PGWT PR wurde ein MPUT gegeben.
87Z	Der PGWT-Aufruf steht im Widerspruch zum Transaktions- oder Vorgangs-Status.
88Z	Ungültige Schnittstellenversion der Datenstruktur (bei KCLI > 0).
89Z	Beim Aufruf der Funktion waren nicht verwendete Parameter nicht mit binär null versorgt.

Eigenschaften des PGWT-Aufrufs

- Wurden Nachrichten oder Nachrichtenteile, die openUTM nach dem INIT für das Programm bereithielt, im Programm mit MGET nicht gelesen, gehen sie verloren.
- PGWT KP und PR

- Ein PGWT KP-Aufruf entspricht einem PEND KP mit anschließenden INIT/INIT PU. PGWT KP ist immer dann erlaubt, wenn ein PEND KP-Aufruf erlaubt ist.
- Ein PGWT PR-Aufruf entspricht einem PEND PA/PR mit anschließenden INIT/INIT PU. PGWT PR ist nur sinnvoll, wenn zuvor ein DGET-Aufruf mit Warten abgesetzt wurde und wenn auf eine Nachricht für die angegebene Queue gewartet werden muss. Nach PGWT PR wartet das Programm, bis eine Nachricht für diese Queue eintrifft.
- PGWT KP und PR bewirken keinen Sicherungspunkt! Die bei PGWT KP durch MPUT veranlassten Dialog-Nachrichten werden ausgegeben. Die restlichen Ausgabeoperationen LPUT, FPUT, SPUT, PTDA und die Freigabeaktion SREL bleiben bis zum nächsten Sicherungspunkt gespeichert.
- PGWT KP und PR sperren Betriebsmittel (LSSBs, GSSBs, TLS, ULS, ggf. Bereiche in Datenbanken) über einen Verarbeitungsschritt hinaus. Aufrufe aus anderen Transaktionen (SGET, SPUT, SREL, PTDA oder GTDA), die auf diese

  Betriebsmittel zugreifen wollen, werden mit einem Returncode (40Z und KCRCDC-Code) abgewiesen. Deshalb wird empfohlen, bei Verwendung des PGWT KP/PR die globalen Betriebsmittel erst vor der Nutzung zu belegen und sie anschließend sofort wieder frei zu geben.
- Wurde vor einem PGWT KP oder PR eine DB-Transaktion begonnen, wird diese Transaktion nicht abgeschlossen! Erst ein PEND RE, SP, FI, FC oder ein PGWT CM führen zum Transaktionsende. RSET, PEND, RS FR, ER oder PGWT RB setzen die Transaktion zurück.
- Wenn die Transaktion zurückgesetzt wird, geht eine durch einen PGWT KP auszuführende MPUT-Ausgabe verloren. Der Vorgang wird auf den letzten Sicherungspunkt zurückgesetzt. Es wird sofort ein Vorgangswiederanlauf durchgeführt, sofern der Benutzer noch angemeldet ist.
  Wenn der Benutzer abgemeldet wurde, z.B. weil die Verbindung zum Client verloren ging, wird beim Anmelden ein Vorgangswiederanlauf durchgeführt, sofern die Benutzerkennung (in Anwendungen ohne Benutzerkennungen der LTERM-Partner) mit Wiederanlauf-Eigenschaft generiert ist (Generierungs-Operanden RESTART=YES der LTERM bzw. USER-Anweisung, siehe openUTM-Handbuch „Anwendungen generieren“).
  Nach einem Anwendungsende ist in stand-alone UTM-Anwendungen ein Vorgangswiederanlauf nur in UTM-S-Anwendungen möglich.

- PGWT CM und RB
  - PGWT CM mit MPUT-Nachricht ist immer dann erlaubt, wenn auch ein PEND RE erlaubt ist. PGWT CM entspricht dann PEND RE mit anschließendem INIT/INIT PU.
  - PGWT CM ohne MPUT-Nachricht ist immer dann erlaubt, wenn auch ein PEND SP erlaubt ist. PGWT CM entspricht dann PEND SP mit anschließendem INIT/INIT PU.
  - Mit PGWT CM wird ein Sicherungspunkt gesetzt, wobei der Programmkontext erhalten bleibt. PGWT CM verhält sich in Bezug auf Datenbank-Transaktionen wie ein PEND SP oder PEND RE.
  - PGWT CM und PGWT RB sind nicht erlaubt, wenn in der aktuellen Transaktion mit einem Partner über das LU6.1-Protokoll kommuniziert wird.
  - Mit PGWT RB wird die Transaktion zurückgesetzt. Der Programmkontext bleibt Im Gegensatz zu RSET oder PEND RS erhalten. Dazu gehören z.B. KB-Programmbereich, SPAB und lokale Datenbereiche.
  - Vor PGWT RB darf keine Rücksetznachricht erzeugt werden.
  - Auf einen mit PGWT CM gesetzten Sicherungspunkt ist kein Vorgangs-Wiederanlauf möglich. Deshalb kann die Folge-Transaktion nur mit PGWT RB zurückgesetzt werden, ohne den abnormal Vorgang zu beenden.
- Wenn der PGWT-Aufruf nicht erfolgreich ausgeführt werden kann, ruft openUTM intern PEND ER auf.
- Fortsetzen der Verarbeitung

- Nach einem PGWT-KP-Aufruf oder einem PGWT-CM-Aufruf mit MPUT wird in das aufrufende Programm zurückgekehrt, sobald alle Antworten vorliegen.
- Nach einem PGWT CM ohne MPUT und einem PGWT RB wird die Verarbeitung sofort fortgesetzt, Ausnahme siehe Verteilte Verarbeitung mit OSI TP auf "PGWT Wartepunkt im Programm setzen ohne Teilprogrammende".
- Nach einem PGWT PR-Aufruf wird die Verarbeitung fortgesetzt, sobald eine Nachricht auf der Queue eingetroffen ist. Um eine solche Nachricht zu lesen, muss ein DGET-Aufruf verwendet werden.

> **i** Verwenden Sie den Aufruf sparsam, da hiermit der Prozess belegt bleibt und während der Wartezeit keine anderen Aufgaben bearbeiten kann. Dies gilt besonders, wenn der PGWT-Aufruf auf eine Eingabe von einem Terminal wartet. Dann ist der Prozess solange blockiert, bis eine Eingabe über die Tastatur erfolgt!

PGWT-Aufruf in einem verteilten OSI TP-Vorgang mit Commit

Wurde bei Verteilter Verarbeitung über OSI TP die Commit-Funktionalität ausgewählt, dann führen PGWT CM ohne MPUT und PGWT RB **immer** zu einem Wartepunkt, d.h.:

- bei PGWT CM ohne MPUT wird die Verarbeitung erst fortgesetzt, nachdem das Transaktionsende von allen Partnern bestätigt oder die Transaktion zurückgesetzt wurde, z.B. wegen eines Fehlers.
- bei PGWT RB wird die Verarbeitung erst fortgesetzt, nachdem das Rücksetzen von allen Partnern bestätigt wurde.

Es wird auch dann ins Teilprogramm zurückgekehrt, wenn

- bei einem PGWT KP Aufruf eine Situation erkannt wurde, die ein Vorsetzen der Transaktion nicht erlaubt.
- bei einem PGWT CM oder RB die aktuelle Transaktion vorgesetzt bzw. zurückgesetzt wurde und eine Situation vorliegt, die ein Vorsetzen der aktuellen Transaktion oder der Folgetransaktion nicht erlaubt.

Diese Situation wird dem Teilprogramm durch das Feld KCTARB im KB-Kopf angezeigt:

KCTARB

zeigt in einem OSI TP-Vorgang an, ob eine Situation eingetreten ist, die ein Rücksetzen der Transaktion erfordert.

Leerzeichen	es ist keine Situation eingetreten, die ein Rücksetzen der Transaktion erfordert.
Y	es ist eine Situation eingetreten, die ein Vorsetzen der Transaktion nicht erlaubt. Es ist noch Kommunikation mit den Partner-Vorgängen erlaubt. Ein Aufruf zum Vorsetzen der Transaktion führt zum abnormalen Vorgangsende.

7.20 PTDA Schreiben in einen TLS

Mit dem Aufruf PTDA (put data) schreiben Sie einen Block aus einem angegebenen Speicherbereich in einen Terminal-spezifischen Langzeitspeicher (TLS) eines LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners.

Ein Teilprogrammlauf eines Dialog-Vorgangs kann nur Blöcke des "eigenen" TLS schreiben, d.h. Blöcke des LTERM/LPAP/OSI-LPAP-Partners, über den der Vorgang gestartet wurde.

Ein Teilprogrammlauf eines Asynchron-Vorgangs kann die Blöcke jedes LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners der UTM-Anwendung beschreiben.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**			
	KCOP	**KCLA**	**KCRN**	**KCLT**
Schreiben in einen TLS-Block (im Dialog-Programm)	"PTDA"	Länge	Blockname	—
Schreiben in einen TLS-Block (im Asynchron-Programm)	"PTDA"	Länge	Blockname	LTERM/LPAP/OSI-LPAP /Master-LPAP-Name

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, der die zu schreibende Nachricht enthält.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"PTDA"
KCLA	Länge in Byte
KCRN	Blockname
KCLT	Name des LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_PTDA	(nb,kcla,kcrn,kclt)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den PTDA-Aufruf ein:

KCOP

im Feld KCOP den Operationscode "PTDA".

KCLA

im Feld KCLA die Länge, in der openUTM die Daten in den TLS schreiben soll. Die hier angegebene Länge wird zur neuen Länge des TLS-Blocks.

KCRN

im Feld KCRN den Namen des TLS-Blocks, in den openUTM die Daten schreiben soll.

KCLT

nur bei Asynchron-Programmen:
im Feld KCLT den Namen des LTERM/LPAP/OSI-LPAP/Master-LPAP-Partners, in dessen TLS openUTM schreiben soll (von Dialog-Programmen wird dieses Feld nicht ausgewertet).

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Nachricht ein, die Sie in den TLS schreiben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: die Adresse des KDCS-Parameterbereichs.

2. Parameter

als 2. Parameter: die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe unten.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim PTDA-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout), siehe KCRCDC.
41Z	Der Aufruf wurde im ersten Teil des Anmelde-Vorgangs gegeben, obwohl dies in der Generierung nicht erlaubt wurde.
43Z	Die Längenangabe in KCLA ist negativ bzw. ungültig.
44Z	Der Name des Blocks in KCRN ist unbekannt oder ungültig.
46Z	Der LTERM/LPAP/OSI-LPAP/Master-LPAP-Name in KCLT ist ungültig (nur bei Asynchron-Programmen).
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Programm wurde kein INIT gegeben.

Eigenschaften des PTDA-Aufrufs

- Bei Transaktionsende (PEND RE/FI/FC/SP) wird die Änderung des TLS-Blocks durchgeführt und der Block entsperrt. Andere Transaktionen können ihn dann wieder verwenden.
  Bei PEND RS/ER/FR oder RSET werden die Änderungen der TLS-Blöcke rückgängig gemacht und die Blöcke entsperrt.
- Die Sperre wird über evtl. über einen längeren Zeitraum hinaus beibehalten bei
  - PEND KP und PGWT KP
  - PEND PA/PR mit Taskwechsel wegen TAC-Klassensteuerung
  - PEND PA/PR mit Warten auf eine DGET-Nachricht.
- Ein PTDA-Aufruf sperrt den Zugriff auf einen TLS-Block bis zum nächsten Sicherungspunkt. Alle anderen TLS-Blöcke des angesprochenen LTERM-/ LPAP- oder OSI-LPAP-Partners sind nicht gesperrt.

Beachten Sie, dass ein TLS-Block aktuell in der Länge existiert, in der er beim letzten PTDA-Aufruf geschrieben wurde.

Wie openUTM reagiert, wenn der gewünschte TLS-Block gesperrt ist, ist in Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“ beschrieben.

7.21 QCRE Temporäre Queue erzeugen

Mit dem Aufruf QCRE (queue create) wird dynamisch eine Temporäre Queue erzeugt.

Voraussetzung für einen erfolgreichen QCRE-Aufruf ist, dass bei der Generierung genügend viele Tabellenplätze für QUEUE-Objekte mit der QUEUE-Anweisung reserviert wurden.

Im QCRE-Aufruf können Sie einen Namen für die zu erzeugende Queue vergeben oder festlegen, dass openUTM automatisch einen Namen vergibt, der dann im Rückgabefeld KCRQN (Queue-Name) eingetragen wird. openUTM erzeugt aufeinanderfolgende Queue-Namen aus abdruckbaren Ziffern. Werden die Queue-Namen von openUTM vergeben, so werden Queue-Namen erst nach 100 Millionen QCRE-Aufrufen erneut verwendet. Damit ist sichergestellt, dass langlaufende Vorgänge für die Kommunikation nicht unbeabsichtigt eine Temporäre Queue verwenden, deren Name nach zwischenzeitlichem Löschen neu vergeben wurde.

Im Folgenden wird das Format des QCRE-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Message Queuing" finden Sie in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich					
	KCOP	KCOM	KCRN	KCLA	KCMF /kcfn	KCQMODE
Queue ohne Namensangabe erzeugen	"QCRE"	"NN"	Leerzeichen	Queue-Level	Leerzeichen	"S" / "W" / binär null
Queue mit Namensangabe erzeugen	"QCRE"	"WN"	Queue-Name	Queue-Level	Leerzeichen	"S" / "W" / binär null

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"QCRE"
KCOM	"NN"/ "WN"
KCRN	Leerzeichen / Name der Queue
KCLA	Queue Level der Queue
KCMF / kcfn	Leerzeichen
KCQMODE	"S" / "W" / binär null

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_QCRENN	(kcla,qmode)
KDCS_QCREWN	(kcla,kcrn,qmode)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRQN	von openUTM vergebener Name
KCRCCC	Returncode

Im KDCS-Parameterbereich machen Sie für den QCRE-Aufruf folgende Angaben:

KCOP

Im Feld KCOP tragen Sie den Operationscode QCRE ein.

KCOM

Im Feld KCOM**:**

- NN (no name),
  wenn openUTM den Namen der Queue automatisch erzeugen soll
- WN (with name),
  wenn Sie den Namen selbst vergeben

KCRN

Im Feld KCRN tragen Sie den Namen der Queue ein (KCOM=WN) bzw. Leerzeichen (KCOM=NN). Ein von Ihnen vergebener Name darf nicht mit einer Ziffer beginnen und muss den Konventionen für generierbare Namen genügen. D.h. er darf nur aus den Zeichen A...Z, a...z, 0...9, $, #, @ bestehen und muss ggf. mit Leerzeichen aufgefüllt werden.

KCLA

Im Feld KCLA geben Sie das Queue-Level an, das heißt, Sie geben an, wieviele Nachrichten in dieser Queue maximal gespeichert werden können.
Wenn Sie Null angeben, verwendet openUTM den Wert bzw. Standardwert des Parameters QLEV aus der QUEUE-Anweisung der Generierung.

KCMF / kcfn

Das Feld KCMF/kcfn muss mit Leerzeichen versorgt werden.

KCQMODE

Im Feld KCQMODE:

- S (Standard),
  wenn beim Erreichen des Queue-Levels weitere Nachrichten abgewiesen werden sollen

- W (wrap),
  wenn beim Erreichen des Queue-Levels eine neue Nachricht die älteste vorhandene Nachricht überschreibt
- binär null:
  openUTM verwendet den Wert bzw. Standardwert des Parameters QMODE aus der QUEUE-Anweisung der Generierung.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRQN

im Feld KCRQN den automatisch erzeugten Namen (bei KCOM=NN).

KCRCCC

im Feld KCRCCC den KDCS-Returncode (siehe nächste Seite).

KDCS-Returncodes im Feld KCRCCC beim QCRE-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde durchgeführt.

16Z KCOM=WN: Der Queue-Name existiert bereits

KCOM=NN: openUTM hat keinen freien Namen gefunden. In diesem Fall können Sie versuchen, mit einem weiteren Aufruf QCRE NN einen freien Queue-Namen zu finden.

Hinweis zur Namensvergabe:
openUTM vergibt Namen, die nur aus Ziffern bestehen und merkt sich jeweils den letzten vergebenen Namen. Bei QCRE mit KCOM=NN durchsucht openUTM die nächsten 100 aufeinander folgenden, aus Ziffern bestehenden Namen nach einem freien Eintrag. Sind diese Namen alle belegt, dann bricht openUTM die Suche mit 16Z ab. Beim nächsten Aufruf QCRE NN werden dann die nächsten 100 Namen durchsucht.

40Z Die Operation kann nicht ausgeführt werden, weil:

- kein freier Tabellenplatz (mehr) für Temporäre Queues vorhanden ist

  (Maßnahme: Wert für NUMBER in der QUEUE-Anweisung vergrößern und neu generieren oder nicht mehr benötigte Temporäre Queues mit QREL löschen).
- im Prozess-spezifischen Puffer für Wiederanlaufdaten kein Platz mehr ist

  (Maßnahme: Wert von MAX RECBUF=(..., *length*) vergrößern und neu generieren).

42Z Der Wert in KCOM ist ungültig.

43Z Der Wert in KCLA (Queue Level) ist negativ bzw. ungültig.

44Z Der Queue-Name beginnt mit einer Ziffer (KCOM=WN) bzw. KCRN enthält keine Leerzeichen (KCOM=NN).

45Z KCMF/kcfn wurde nicht mit Leerzeichen versorgt.

46Z Der Wert in KCQMODE ist ungültig.

49Z Nicht verwendete Felder haben einen Wert ungleich binär null.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z Im Teilprogrammlauf wurde noch kein INIT aufgerufen.

Eigenschaften des QCRE-Aufrufs

- Zum Erzeugen einer Temporären Queue ist keine Administrationsberechtigung erforderlich.
- Werden die Queue-Namen von openUTM vergeben, so werden diese erst nach 100 Millionen QCRE-Aufrufen erneut verwendet.
- Wurde eine Temporäre Queue mit QCRE neu erzeugt, dann können schon in der selben Transaktion Nachrichten in diese Queue geschrieben werden. Diese Nachrichten lassen sich jedoch erst nach dem erfolgreichen Abschluss der Transaktion lesen und administrieren.
- Bei UTM-S bleiben Temporäre Queues und ihre Nachrichten über den Anwendungslauf hinaus erhalten, bis sie explizit mit einem QREL-Aufruf gelöscht werden. Bei UTM-F werden Temporäre Queues automatisch mit dem Ende des Anwendungslaufs gelöscht. Alle noch in der Queue gespeicherten Nachrichten gehen verloren.

7.22 QREL Temporäre Queue löschen

Mit dem Aufruf QREL(queue release) wird dynamisch eine Temporäre Queue gelöscht. Dabei werden alle Nachrichten der Queue gelöscht und der Name sowie der Tabellenplatz der Queue freigegeben.

Vorgänge, die auf DGET-Nachrichten dieser Queue warten, werden fortgesetzt.

Im Folgenden wird das Format des QREL-Aufrufs ausführlich dargestellt. Weitere Informationen zum Thema "Message Queuing" finden Sie in Abschnitt „Message Queuing (Asynchron-Verarbeitung)“.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich			
	KCOP	KCOM	KCRN	KCMF/kcfn
Temporäre Queue löschen	"QREL"	"RL"	Name der Queue	Leerzeichen

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"QREL"
KCOM	"RL"
KCRN	Name der Queue
KCMF/kcfn	Leerzeichen

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_QRELRL	(kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode

Im KDCS-Parameterbereich machen Sie für den QREL-Aufruf folgende Angaben:

KCOP

Im Feld KCOP tragen Sie den Operationscode QREL ein.

KCOM

Im Feld KCOM tragen Sie die Modifikation RL ein.

KCRN

Im Feld KCRN tragen Sie den Namen der zu löschenden Queue ein.

KCMF/kcfn

Das Feld KCMF/kcfn muss mit Leerzeichen versorgt werden.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode (siehe unten).

KDCS-Returncodes im Feld KCRCCC beim QREL-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde durchgeführt.
40Z	Es ist kein Platz mehr im Prozess-spezifischen Puffer für Wiederanlaufdaten. Zu Engpässen kann es kommen, weil openUTM für jede Nachricht, die in einer noch nicht abgeschlossenen Transaktion mittels DGET gelesen wird, einen eigenen DADM-Aufruf durchführt und ein "processing item" in den Puffer schreiben muss. Maßnahme: Wert von MAX RECBUF=(..., *length*) vergrößern und neu generieren.
42Z	Der Wert in KCOM ist ungültig.
44Z	Es existiert keine Temporäre Queue mit dem in KCRN angegebenen Namen.
45Z	KCMF/kcfn wurde nicht mit Leerzeichen versorgt.
49Z	Nicht verwendete Felder (außer KCMF) haben einen Wert ungleich binär null.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	Im Teilprogrammlauf wurde noch kein INIT aufgerufen.

Eigenschaften des QREL-Aufrufs

- Zum Löschen einer Temporären Queue ist keine Administrationsberechtigung erforderlich.
- Nach dem Aufruf QREL können keine Nachrichten der gelöschten Queue mehr gelesen oder administriert werden. Neue Nachrichten für diese Queue können nicht erzeugt werden.
- Nach einem QREL-Aufruf und erfolgreichem Abschluss der Transaktion kann eine neue Temporäre Queue mit demselben Namen erzeugt werden.

7.23 RSET Transaktion zurücksetzen

Mit dem Aufruf RSET (reset transaction) können Sie Änderungen und Operationen der Transaktion rückgängig machen. Auch offene Datenbank-Transaktionen werden zurückgesetzt. Alle Ausgabe-Operationen seit dem letzten lokalen Sicherungspunkt werden verworfen. Die Kontrolle wird an das Teilprogramm zurückgegeben: Das Teilprogramm wird hinter dem RSET-Aufruf fortgesetzt. Anschließend sind weitere KDCS-Aufrufe (außer INIT) sowie Datenbank-Aufrufe möglich.

Mit dem RSET-Aufruf lässt sich auf Anwendungsfehler mit gezielten Aktionen reagieren. Sie können eine Transaktion zurücksetzen und gleichzeitig die Kontrolle im Anwendungsprogramm zurückerhalten.

Sinnvoll ist ein solches Vorgehen bei Fehlern, die keine Programmfehler sind (z.B. als Reaktion auf KDCS-Returncodes >= 40Z). Im Teilprogramm können Sie dann gezielt reagieren, z.B. durch:

- Senden einer Nachricht an den betreffenden Client oder an den Administrator (MPUT)
- Schreiben einer Logging-Information (LPUT)
- Senden eines Ausgabeauftrags, z.B. an einen Drucker (FPUT/DPUT)

Der RSET-Aufruf kann z.B. auch dann sinnvoll sein, wenn Datenbankzugriffe unerwartete Returncodes liefern (z.B. "Satz nicht vorhanden") und bereits UPDATE-Operationen gelaufen sind.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Für den RSET-Aufruf ist nur die Angabe des Operationscodes "RSET" im Feld KCOP notwendig.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"RSET"

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++-Makroaufruf	
Makronamen	Parameter
KDCS_RSET	()

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den RSET-Aufruf ein:

KCOP

im Feld KCOP den Operationscode RSET.

Andere Operanden des Parameterbereiches wertet openUTM nicht aus.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim RSET-Aufruf

Im Programm ist auswertbar:

000	Die Operation wurde ausgeführt.

Weitere Returncodes sind dem Dump zu entnehmen:

70Z	Die Operation kann vom System nicht ausgeführt werden (Generierungs- bzw. Systemfehler), siehe KCRCDC.
71Z	Es wurde noch kein INIT gegeben.

Eigenschaften des RSET-Aufrufs

- Alle bis zum RSET-Aufruf von dieser Transaktion belegten Betriebsmittel werden freigegeben.
- Alle Vorgangs-spezifischen Daten werden auf den letzten Sicherungspunkt zurückgesetzt, d.h. es stehen der KB-Programmbereich, alle LSSBs und GSSBs sowie TLS- und ULS-Blöcke mit ihren alten Inhalten wieder zur Verfügung.
- Daten im SPAB und in Programm-spezifischen Arbeitsspeicherbereichen bleiben unverändert.
- Eine offene DB-Transaktion wird zurückgesetzt.
- Jedes Rücksetzen einer DB-Transaktion hat die gleiche Wirkung wie ein RSET-Aufruf, führt also implizit zum Rücksetzen der UTM-Transaktion.

- Das Teilprogramm erhält die Kontrolle nach dem RSET-Aufruf zurück und setzt den Programmlauf mit der nächsten Anweisung fort, welche auf den RSET-Aufruf folgt. Im Teilprogramm können dann weitere KDCS-Aufrufe (mit Ausnahme von INIT) und DB-Aufrufe gegeben werden.
- Eine vor dem RSET-Aufruf gelesene Dialog-Eingabenachricht lässt sich anschließend nicht mehr lesen. Eine vor einem RSET-Aufruf noch nicht gelesene Dialog-Eingabenachricht lässt sich anschließend noch lesen.
- Mit FGET oder DGET gelesene Eingabe-Nachrichten lassen sich nach einem RSET wieder lesen. Bei DGET-Nachrichten jedoch nur dann, wenn die in der Generierung festgelegte maximale Anzahl erneuter Zustellungen ist noch nicht erreicht. Näheres siehe openUTM-Handbuch „Anwendungen generieren“, Operand REDELIVERY in der MAX-Anweisung.

  Ist die maximale Anzahl erneuter Zustellung erreicht, so wird die Nachricht gelöscht oder von UTM in der Dead Letter Queue gesichert (nur bei Nachrichten an eine TAC-Queue möglich), siehe openUTM-Handbuch „Anwendungen generieren“, Operand DEAD-LETTER-Q in der TAC-Anweisung.

Eigenschaften des RSET-Aufrufs bei verteilter Verarbeitung

Das Verhalten von openUTM nach einem RSET-Aufruf in einem Teilprogrammlauf, der zu einer verteilten Transaktion gehört, ist abhängig vom Generierungsparameter RSET der UTMD-Anweisung (siehe openUTM-Handbuch „Anwendungen generieren“):

- Ist RSET=LOCAL generiert, dann hat der RSET-Aufruf keine Auswirkungen auf die verteilte Transaktion. Dabei kann es zu Inkonsistenzen in den verteilten Datenbeständen kommen, wenn einige der an der verteilten Transaktion beteiligten lokalen Transaktionen vorgesetzt und andere zurückgesetzt werden. Bei dieser Generierung wird die globale Datenkonsistenz nicht mehr von den beteiligten Systemkomponenten garantiert, sondern liegt in der Verantwortung der Anwendungsteilprogramme. Diese müssen entscheiden, in welchen Situationen die verteilte Transaktion noch sinnvoll beendet werden kann und in welchen sie zurückgesetzt werden muss.
- Ist RSET=GLOBAL generiert, dann erzwingt openUTM, dass der Teilprogrammlauf mit einer PEND-Variante beendet wird, die zum Rücksetzen der verteilten Transaktion führt (siehe auch Abschnitt "PEND Beenden eines Teilprogramms").

7.24 SGET Lesen aus einem Sekundärspeicherbereich

Mit dem Aufruf SGET (storage GET) lesen Sie Daten von einem Sekundärspeicherbereich in einen Speicherbereich des Teilprogramms. Als Sekundärspeicherbereiche kommen dabei infrage:

- der Globale Sekundäre Speicherbereich (GSSB)
- der Lokale Sekundäre Speicherbereich (LSSB)
- der User-spezifische Langzeitspeicher (ULS)

Falls ein LSSB nicht mehr benötigt wird, kann er durch Angabe von KCOM=RL gleichzeitig gelöscht werden. Der Inhalt eines ULS kann nur durch Schreiben (SPUT) mit KCLA=0 gelöscht werden.

Ein GSSB muss mit einem eigenen Aufruf (SREL) gelöscht werden; er ist bis zum Ende der Transaktion bzw. des Vorgangs gesperrt. Siehe hierzu die Beschreibung des SREL-Aufrufs.
Mit dem Aufruf UNLK kann ein GSSB oder ULS explizit entsperrt werden.

Unix-, Linux- and Windows-Systeme
In UTM-Cluster-Anwendungen stehen GSSB oder ULS Cluster-weit zur Verfügung.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**				
	KCOP	**KCOM**	**KCLA**	**KCRN**	**KCUS**
Lesen aus einem LSSB	"SGET"	"KP"	Länge	Name des LSSB	-
Lesen aus einem LSSB und Löschen des LSSB	"SGET"	"RL"	Länge	Name des LSSB	-
Lesen aus einem GSSB (mit Sperren des GSSB)	"SGET"	"GB"	Länge	Name des GSSB	-
Lesen aus einem ULS (mit Sperren des ULS)	"SGET"	"US"	Länge	Blockname	Benutzerkennung/LSES-Name /Association-Name/Leerzeichen

Bei KCOM = US müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, in den openUTM die Nachricht lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"SGET"
KCOM	"KP"/"RL"/"GB"/"US"
KCLA	Länge in Byte
KCRN	Name des Bereichs
KCUS	Benutzerkennung/LSES-Name/Association-Name/Leerzeichen

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_SGETKP/KDCS_SGETRL/KDCS_SGETGB	(nb,kcla,kcrn)
KDCS_SGETUS	(nb,kcla,kcrn,kcus)

Rückgaben von openUTM	
Nachrichtenbereich	**Inhalt**
	Daten
Feldname im KB-Rückgabebereich	
KCRLM	tatsächliche Blocklänge
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den SGET-Aufruf ein:

KCOP

im Feld KCOP den Operationscode SGET.

KCOM

im Feld KCOM:

- KP (keep) zum Lesen eines LSSBs - der Bereich bleibt erhalten
- RL (release) zum Lesen und Löschen eines LSSBs
- GB zum Lesen eines GSSBs
- US zum Lesen eines Blocks eines ULS

KCLA

im Feld KCLA die Länge, in der die Daten in den Nachrichtenbereich übertragen werden sollen.

KCRN

im Feld KCRN den Namen des LSSB/GSSB bzw. des ULS-Blocks, aus dem gelesen werden soll.

KCUS

im Feld KCUS die Benutzerkennung/Session/Association, wenn ein ULS-Block einer fremden Benutzerkennung/Session/Association gelesen werden soll, sonst Leerzeichen (bei Angabe von Leerzeichen wird der ULS-Block des Benutzers/Session/Association gelesen, der den Vorgang gestartet hat).
Wird eine fremde Benutzerkennung/Session/Association in KCUS eingetragen, dann muss die eigene Benutzerkennung/Session/Association administrationsberechtigt sein.

Bei KCOM = KP/RL/GB: irrelevant.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

2. Parameter

als 2. Parameter: die Adresse des Nachrichtenbereichs, in den openUTM die Nachricht einlesen soll. Die Adresse des Nachrichtenbereichs müssen Sie auch dann angeben, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

Nachrichtenbereich

im angegebenen Nachrichtenbereich die gewünschten Daten.

KCRLM

in KCRLM die tatsächliche Länge der Daten im LSSB/GSSB/ULS (in Bytes). Damit können Sie Abweichungen von der Angabe in KCLA feststellen (wichtig, wenn KCLA kleiner angegeben wurde).
Ausnahme: Bei KCLA = 0 wird immer KCRLM = 0 zurückgegeben.

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim SGET-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
14Z	Unter dem in KCRN angegebenen Namen ist kein Bereich vorhanden (nur bei KP, RL, GB).
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout), siehe KCRCDC.
41Z	Der Aufruf wurde im ersten Teil des Anmelde-Vorgangs abgesetzt, obwohl dies in der Generierung nicht erlaubt wurde. bei KCOM=US: der Aufruf wurde im ersten Teil des Anmelde-Vorgangs oder im Anmelde-Vorgang nach einem SIGN ON und vor dem PEND PS Aufruf abgesetzt.
42Z	Der Eintrag in KCOM ist ungültig.
43Z	Die Längenangabe in KCLA ist negativ bzw. ungültig.
44Z	Name in KCRN ungültig. Er ist ungültig, wenn er nur aus Leerzeichen oder nur aus binär null besteht oder nicht generiert ist (bei ULS).
46Z	Die Angabe in KCUS ist ungültig.
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.
49Z	Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null (nur bei KCOM = US).

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Programm wurde kein INIT gegeben.

Eigenschaften des SGET-Aufrufs

- Der Bereich wird in der tatsächlichen Länge übertragen, höchstens jedoch in der bei KCLA angegebenen Länge. Die tatsächliche Länge der Daten im GSSB, LSSB oder ULS wird im Feld KCRLM zurückgegeben:
  - Ist die in KCLA angegebene Länge kleiner als die tatsächliche Länge des zu lesenden Satzes, wird rechts abgeschnitten. Im Teilprogramm kann dieses Situation abgefangen werden (KCLA < KCRM).
  - Ist die in KCLA angegebene Länge größer als die tatsächliche Länge des zu lesenden Satzes (KCLA > KCRLM), ist nach dem SGET-Aufruf der überschüssige Teil des Nachrichtenbereichs undefiniert.
- Wird in einem Teilprogramm versucht, die Leseoperation SGET auf einen nicht vorhandenen Speicherbereich durchzuführen, bekommt das Teilprogramm den Returncode 14Z (kein Bereich mit diesem Namen vorhanden).
- Wird mit einem SGET auf einen GSSB oder ULS zugegriffen, gilt:
  - Ein SGET-Aufruf für einen GSSB bzw. ULS-Block sperrt diesen bis zum nächsten Sicherungs- bzw. Rücksetzpunkt (d.h. bis zum PEND SP/RE/FI/FC/RS/ER/FR oder RSET). Wird der Verarbeitungsschritt nach einem SGET-Aufruf in einem Teilprogramm mit einem PEND KP, PGWT KP, PGWT PR oder mit einem PEND PA/PR mit Taskwechsel wegen TAC-Klassensteuerung oder Wartens auf DGET-Nachricht abgeschlossen, bleibt die Zugriffssperre bis zum nächsten Sicherungspunkt erhalten, außer die Sperre wird vorher durch einen UNLK-Aufruf gelöst.
  - Lesen aus einem nicht vorhandenen GSSB hat die gleiche Wirkung wie Erzeugen eines GSSB mit gleichzeitigem Löschen (SPUT, SREL-Folge), d.h.:
    - Der Name dieses GSSB bleibt bis zum nächsten Sicherungs- oder Rücksetzpunkt gesperrt.
    - Ist die generierte maximale Anzahl von GSSBs bereits erreicht, bekommt daher das Teilprogramm den Returncode 40Z mit KCRCDC K804 zurück.

    Unix-, Linux- and Windows-Systeme
    In einer UTM-Cluster-Anwendung sind für das Lesen aus einem nicht vorhandenen GSSB vier zusätzliche Datei-Zugriffe notwendig (zum Erhöhen und Vermindern des GSSB-Zählers). Daher ist es in UTM-Cluster-Anwendungen aus Performancegründen sinnvoll, etwa zur Serialisierung von Teilprogrammen keinen leeren GSSB, sondern z.B. einen GSSB der Länge 1 zu verwenden.

  Wie openUTM reagiert, wenn der gewünschte GSSB oder ULS-Block gesperrt ist, ist in Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“ beschrieben.
- Wird mit einem SGET auf einen LSSB zugegriffen, gilt:
  - SGET KP bewirkt, dass der LSSB auch noch in der Folgetransaktion des Vorgangs, d.h. nach dem nächsten PEND RE/SP zur Verfügung steht.
  - SGET RL liest den LSSB und löscht ihn bei Transaktionsende (d.h. bei PEND RE/SP/FI/FC/RS/ER/FR). Ein zwischenzeitlicher Zugriffsversuch wird mit 14Z zurückgewiesen.
    Diese Variante sollten Sie immer dann verwenden, wenn der LSSB nach dem Lesen im laufenden Vorgang nicht mehr benötigt wird.

7.25 SIGN An- und Abmelden steuern, Berechtigungsdaten überprüfen, Passwort ändern

Mit dem Aufruf SIGN (sign on) können Sie

- im Anmelde-Vorgang den Status des Anmelde-Vorgangs abfragen oder die Berechtigungsdaten an openUTM übergeben,
- das Passwort für die aktuelle Benutzerkennung ändern,
- Berechtigungsdaten überprüfen lassen
- im Programm die Wirkung der Kommandos KDCOFF und KDCOFF BUT auslösen.

Der SIGN-Aufruf ist nur in Dialog-Teilprogrammen erlaubt (Ausnahme: SIGN CK).

Für B2000-Systeme gibt es zusätzlich den Aufruf SIGN CL (Change Locale) zum Ändern des Locale für die aktuelle Benutzerkennung. Dieser Aufruf ist ab "SIGN CL - Locale der Benutzerkennung ändern (BS2000-Systeme)" beschrieben.

Versorgen des KDCS-Parameterbereichs (1. Parameter)

Die folgende Tabelle zeigt die notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich			
	KCOP	KCOM	KCLA	KCUS
Status des Anmelde-Vorgangs abfragen	"SIGN"	"ST"	120	binär null
Berechtigungsdaten an openUTM übergeben	"SIGN"	"ON"	16	Benutzerkennung
Passwort ändern	"SIGN"	"CP"	32	binär null
Berechtigungsdaten überprüfen (ohne Anmeldung)	"SIGN"	"CK"	16	Benutzerkennung
Wirkung des Kommandos KDCOFF auslösen	"SIGN"	"OF"	0	binär null
Wirkung des Kommandos KDCOFF BUT auslösen	"SIGN"	"OB"	0	binär null

Alle nicht-verwendeten Felder des KDCS-Parameterbereichs müssen mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier müssen Sie die Adresse des Nachrichtenbereichs bereitstellen, aus dem openUTM die Daten lesen soll.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"SIGN"
KCOM	"ST"/"ON"/"CP"/"CK"/"OF"/"OB"
KCLA	120/16/32/16/0/0
KCUS	Benutzerkennung/binär null
Nachrichtenbereich	
	Daten/ -

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_SIGNST/KDCS_SIGNOF/KDCS_SIGNOB	(nb)
KDCS_SIGNON/KDCS_SIGNCK	(nb,kcla,kcus)
KDCS_SIGNSTLA/KDCS_SIGNCP	(nb,kcla)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRSIGN1	Anmeldestatus
KCRSIGN2	Zusatzinformation
KCRUS	Name der Benutzerkennung
KCRCCC	Returncode
KCRCDC	interner Returncode
KCRMF/kcrfn	Formatkennzeichen des Startformats / Leerzeichen
KCRLM	Gültigkeitsdauer des Passwortes

In den KDCS-Parameterbereich tragen Sie für den SIGN-Aufruf ein:

KCOP

im Feld KCOP den Operationscode SIGN.

KCOM

im Feld KCOM

- ST: Status des Anmelde-Vorgangs abfragen
- ON: Berechtigungsdaten prüfen (mit Anmelden)
- CP: Passwort ändern,
- CK: Berechtigungsdaten überprüfen (ohne Anmelden)
- OF: Wirkung des Kommandos KDCOFF auslösen,
- OB: Wirkung des Kommandos KDCOFF BUT auslösen.

KCLA

im Feld KCLA tragen Sie ein:

- 120 bei KCOM=ST:
  Dies ist die Länge des Nachrichtenbereichs, in den openUTM die Information übertragen soll. Für die Strukturierung des Nachrichtenbereichs stellt Ihnen openUTM eine Datenstruktur zur Verfügung, siehe Beschreibung auf "Eigenschaften des SIGN-Aufrufs". Im Header der Datenstruktur geben Sie die Versionsnummer der Struktur an.
- 16 bei KCOM = ON/CK:
  Dies ist die Länge des Passworts, das bei diesem Aufruf im Nachrichtenbereich übergeben wird.
- 32 bei KCOM = CP:
  Dies ist die Länge von altem und neuem Passwort, die bei diesem Aufruf im Nachrichtenbereich übergeben werden.
- 0 bei KCOM = OF/OB

KCUS

im Feld KCUS die Benutzerkennung, wenn mit KCOM = ON/CK Berechtigungsdaten an openUTM übergeben werden sollen. Bei allen übrigen Varianten muss binär null eingetragen werden.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Daten ein, die Sie an openUTM übergeben bzw. von openUTM erhalten wollen.
Bei KCOM = ON/CK wird das Passwort (16 Zeichen), bei KCOM = CP das alte und das neue Passwort (32 Zeichen) bereitgestellt.
Bei KCOM = ST für den Fall KCLA > 0 werden folgende Informationen (maximal 120 Zeichen) ausgetauscht:

- übergeben wird die gewünschte Version der Datenstruktur (2 Zeichen)
- geliefert werden Daten zum Anmelde-Vorgang (z.B. Gültigkeitsdauer Passwort).

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: die Adresse des KDCS-Parameterbereichs.

2. Parameter

als 2. Parameter: die Adresse des Nachrichtenbereichs, aus dem openUTM die Daten lesen soll. Die Adresse des Nachrichtenbereichs geben Sie auch an, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRSIGN1
KCRSIGN2

in den Feldern KCRSIGN1 und KCRSIGN2 Zusatzinformationen (es wird nur bei KCRCCC = 000 etwas eingetragen):

- bei KCOM = ST den aktuellen Stand des Anmeldeverfahrens, siehe Tabelle auf "Rückgaben des SIGN ST-Aufrufs in den Feldern KCRSIGN1 und KCRSIGN2".
- Bei KCOM = CK das Ergebnis der Überprüfung, siehe Tabelle auf "Rückgaben des SIGN CK-Aufrufs in den Feldern KCRSIGN1 und KCRSIGN2".

KCRUS

Das Feld KCRUS enthält den Namen der Benutzerkennung, wenn:

- KCRSIGN1=U, d.h die Anmeldung der Benutzerkennung konnte nicht erfolgreich durchgeführt werden, oder
- KCRSIGN1=I, d.h. die Anmeldung wurde noch nicht erfolgreich beendet, für den Benutzer muss noch ein Zwischendialog durchgeführt werden.

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe "KDCS-Returncodes des SIGN-Aufruf".

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KCRMF/kcrfn

im Feld KCRMF/kcrfn bei KCOM = ST das Formatkennzeichen des Startformats oder Leerzeichen, falls kein Startformat generiert wurde. Bei einer Anwendung mit USER und wenn die Anmeldung erfolgreich war, wird das Kennzeichen des User-spezifischen Startformats zurückgegeben. Ist die Anmeldung noch nicht erfolgreich oder ist die Anwendung ohne USER generiert, dann enthält KCRMF/kcrfn das Kennzeichen des LTERM-spezifischen Startformats.

KCRLM

im Feld KCRLM bei KCOM = ST:

- bei KCRCCC < 40Z die Länge der bei openUTM tatsächlich vorhandenen Information.
- bei KCRCCC >= 40Z wird 0 zurückgegeben.

Rückgaben des SIGN ST-Aufrufs in den Feldern KCRSIGN1 und KCRSIGN2

Bei SIGN ST liefert openUTM folgende Informationen über den aktuellen Stand des Anmeldeverfahrens in den Feldern KCRSIGN1 und KCRSIGN2:

KCRSIGN1	KCRSIGN2	Bedeutung
C	01	Verbindungsaufbau ist erfolgt (wie K002).
	02	Kommando KDCOFF BUT wurde eingegeben (wie K018).
	03	KDCOFF BUT wurde aus dem Programm gegeben.
U	01	Der angegebene USER ist nicht generiert (wie K004).
	02	Der angegebene USER ist gesperrt (wie K005).
	03	Mit diesem USER ist bereits jemand angemeldet (wie K007.
	04	Das angegebene bisherige Passwort ist falsch (wie K006)
	05	Angaben zum neuen Passwort nicht verwendbar.
	06	Das Terminal hat keinen Kartenleser (wie K030).
	07	Die Karteninformation ist falsch (wie K031).
	08	Die Anmeldung ist zurzeit nicht möglich wegen Betriebsmittel-Engpass oder es dürfen zurzeit keine weiteren Benutzer mehr angemeldet werden, da die Maximalzahl gleichzeitig angemeldeter Benutzer bereits erreicht ist, oder das Passwort konnte nicht geändert werden, da gerade ein inverser KDCDEF läuft.
	09	Nur auf BS2000-Systemen: Eine Anmeldung ist wegen fehlender Kerberos-Unterstützung nicht möglich (wie K110).
	10	Das aktuelle LTERM hat nicht die Berechtigung, den Vorgang fortzusetzen (wie K123).
	11	Die Gültigkeitsdauer des Passwortes wurde überschritten (wie K120).
	12	Das neue Passwort erfüllt nicht die Anforderungen der generierten Komplexitätsstufe (wie K097).
	13	Das neue Passwort ist zu kurz (wie K097).
	14	Das von KDCUPD übertragene Passwort erfüllt nicht die Anforderung der generierten Komplexitätsstufe (wie K125).

KCRSIGN1	KCRSIGN2	Bedeutung
U	15	Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf erforderlich (wie K145).
	16	Der offene Vorgang kann von diesem LTERM-Partner aus (wie K123) nicht fortgesetzt werden.
	17	Vom Administrator wurde SHUT WARN gegeben; Anmelden (wie K016) an die Anwendung ist für normale Benutzer nicht mehr möglich; ein Administrator darf sich noch anmelden.
	18	Auf der Verbindung ist der für die Fortsetzung des offenen Vorgang (wie K123) nötige Verschlüsselungsmechanismus nicht verfügbar
	19	Die Gültigkeitsdauer des Passwortes wurde überschritten, aber das Passwort darf noch geändert werden, da die Anwendung mit SIGNON GRACE=YES generiert ist.
	20	Nur auf BS2000-Systemen: Bei der Kerberos-Authentisierung ist ein Fehler aufgetreten (wie K108)
	21	Nur auf BS2000-Systemen: Der Kerberos-Principal ist ungültig (wie K109)
	22	Nur auf Unix-, Linux- und Windows-Systemen: Der angegebene USER existiert nicht in der Cluster-User-Datei (wie K004).
	23	Nur auf Unix-, Linux- und Windows-Systemen: Mit diesem USER hat sich bereits jemand an einer anderen Knoten-Anwendung angemeldet (wie K007).
	24	Nur auf Unix-, Linux- und Windows-Systemen: Anmeldung ist zur Zeit nicht möglich, weil die Cluster-User-Datei innerhalb der generierten Zeit (Anweisung CLUSTER, Parameter FILE-LOCK-TIMER-SEC, Parameter FILE-LOCK-RETRY) nicht gesperrt werden konnte (wie K091).
	25	Nur auf Unix-, Linux- und Windows-Systemen: Anmeldung an dieser Knoten-Anwendung nicht möglich, weil ein an eine andere Knoten-Anwendung gebundener Vorgang des Benutzers existiert, der nicht beendet werden darf (wie K189).
	26	Anmeldung abgelehnt, da der offene Vorgang des Benutzers eine Transaktion im Zustand PTC hat, aber kein Vorgangs-Wiederanlauf angefordert wurde.

KCRSIGN1	KCRSIGN2	Bedeutung
I	01	USER ist bekannt, aber es ist noch ein Zwischendialog erforderlich (nur bei automatischem KDCSIGN).
A	01	Anmeldung erfolgreich, da ohne USER generiert (wie K001).
	02	Anmeldung erfolgreich (wie K008).
	03	Nur auf BS2000-Systemen: Anmeldung erfolgreich (über Verteiler).
	04	Anmeldung erfolgreich (von der Verbindungs-Benutzerkennung, nur bei TS-Anwendungen oder UPIC-Client). Mit dem Aufruf SIGN ON ist eine Anmeldung mit einer "echten" Benutzerkennung möglich.
	05	Anmeldung erfolgreich, das Passwort wurde per Zwischendialog geändert.
	06	Nur auf BS2000-Systemen: Anmeldung erfolgreich (über Verteiler), das Passwort wurde über Verteiler geändert.
R	01	wie A01, mit Vorgangs-Wiederanlauf.
	02	wie A02, mit Vorgangs-Wiederanlauf.
	03	wie A03, mit Vorgangs-Wiederanlauf (nur auf BS2000-Systemen).
	04	wie A04, mit Vorgangs-Wiederanlauf.
	05	wie A05, mit Vorgangs-Wiederanlauf.
	06	wie A06, mit Vorgangs-Wiederanlauf (nur auf BS2000-Systemen).

KCRSIGN1 liefert eine Grob-Klassifizierung

- C (Connected) nicht angemeldet.
  Zustand nach Verbindungsaufbau oder KDCOFF BUT.
- U (Signon Unsuccessful) nicht angemeldet.
  Ein vorausgehender Anmeldeversuch wurde abgelehnt.
- I (Signon Incomplete) Anmeldung unvollständig.
  Zwischendialog für Zusatzdaten (Passwort, Ausweis, Chipkarte) ist erforderlich.
- A (Signon Accepted)
  Anmeldung erfolgreich. Ohne nachfolgenden Vorgangs-Wiederanlauf.
- R (Signon Accepted + Restart)
  Anmeldung erfolgreich. Mit nachfolgendem Vorgangs-Wiederanlauf.

Rückgaben des SIGN CK-Aufrufs in den Feldern KCRSIGN1 und KCRSIGN2

Bei SIGN CK liefert openUTM folgende Informationen in den Feldern KCRSIGN1 und KCRSIGN2:

KCRSIGN1	KCRSIGN2	Bedeutung
A	02	Die Berechtigungsdaten sind korrekt und vollständig
U	01	Der angegebene USER existiert nicht
	02	Der angegebene USER ist gesperrt
	04	Das angegebene bisherige Passwort ist falsch
	06	Das Terminal hat keinen Kartenleser (nur auf BS2000-Systemen)
	07	Die Karteninformation ist ungültig (nur auf BS2000-Systemen)
	09	Keine Kerberos-Unterstützung (nur auf BS2000-Systemen)
	11	Die Gültigkeitsdauer des Passwortes wurde überschritten
	14	Das von KDCUPD übertragene Passwort erfüllt nicht die Anforderung der generierten Komplexitätsstufe oder es ist zu kurz
	21	Der Kerberos-Principal ist ungültig (nur auf BS2000-Systemen)

KDCS-Returncodes im Feld KCRCCC beim SIGN CK/CP/OB/ON/OF/ST-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde ausgeführt.

01Z Bei SIGN ST: Die Funktion wurde ausgeführt, der bereitgestellte Nachrichtenbereich ist jedoch zu kurz (Wert in KCLA zu klein). Es wurde keine oder nur unvollständige Information zurückgegeben.

40Z Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler).

41Z Der Aufruf ist an dieser Stelle nicht erlaubt:

- Es wurde zuvor schon ein SIGN OF/OB-Aufruf gegeben oder
- SIGN-Aufruf in einem Asynchron-Vorgang und Aufruf ist nicht SIGN CK oder
- SIGN ST/ON-Aufruf außerhalb eines Anmelde-Vorgangs oder
- SIGN CP/CK-Aufruf vor erfolgreicher Anmeldung oder
- SIGN ON/CP/CK-Aufruf in einer Anwendung, die ohne Benutzerkennungen generiert wurde oder
- SIGN OB/OF in einem Auftragnehmer-Vorgang (bei Verteilter Transaktionsverarbeitung).

42Z Der Eintrag in KCOM ist ungültig.

43Z Die Längenangabe in KCLA ist negativ bzw. ungültig.

44Z Bei KCOM=CP: die Angabe zum alten Passwort stimmt nicht, das Passwort wird nicht geändert,

45Z Bei KCOM=CP: die Angabe zum neuen Passwort stimmt nicht, das Passwort wird nicht geändert. Die genauere Ursache liefert KCRCDC.

47Z Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.

48Z Bei KCOM=ST: ungültige Schnittstellenversion

49Z Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In dem Teilprogrammlauf wurde noch kein INIT-Aufruf gegeben.

Eigenschaften des SIGN-Aufrufs

- Nachrichtenbereich bei SIGN ST mit KCLA > 0:

Feldname COBOL	Feldname C/C++	Länge in Byte	Beschreibung
Aufruf-Information:			
KCVER	if_version	2	Versionsnummer der Datenstruktur (4)
Rückgabe-Information:			
KCRPWVAL	rpwval	2	Gültigkeitsdauer Passwort
KCRPWMIN	rminpw	2	minimale Gültigkeitsdauer Passwort
KCRUSER	ruser	8	Benutzerkennung
KCRTAC	rtac	8	Transaktionscode aus UPIC-Protokoll
KCFILLER	filler	8	
KCLSTSGN	rlstsgn	14	Datum/Uhrzeit des letzten Sign-On
KCDSPMSG	rdispmsg	1	Nachricht vorhanden (Y/N)
KCTAPTC	rtainptc	1	Transaktion im Zustand PTC (Y/N)
KCCLNODE	rclusternode	64	Nur auf Unix-, Linux- und Windows-Systemen: Rechnername des Knotens, an den der offene Vorgang gebunden ist
KCRPASSL	rpsword16	16	Passwort aus UPIC-Protokoll
KCSGRES	reserved	10	reserviert für Erweiterungen

Dabei bedeuten:

KCVER

Versionsnummer der Datenstruktur. In dieser Version von openUTM ist hier 4 anzugeben.

KCRPWVAL

Bei erfolgreicher Anmeldung (KCRSIGN1 = A/R) oder noch nicht erfolgreicher Anmeldung (KCRSIGN1 = I) enthält das Feld die Anzahl Tage, die das Passwort dieses Benutzers noch gültig ist.
Der Wert -1 bedeutet, dass keine Gültigkeitsdauer des Passwortes generiert ist.
Der Wert 0 bedeutet, dass das Passwort innerhalb der nächsten 24 Stunden ungültig wird.
Der Wert -2 bedeutet, dass die Gültigkeit des Passwortes abgelaufen ist. Das Passwort muss geändert werden, wenn der Anmelde-Vorgang erfolgreich beendet werden soll. Dieser Wert ist nur möglich, wenn Grace-Sign-On erlaubt ist (SIGNON-Anweisung der Generierung, Parameter GRACE = YES)

Bei KCRSIGN1 = I bedeutet der Wert -3, dass die Komplexitätsstufe oder die Mindestlänge des Passworts erhöht wurde und das mit dem Tool KDCUPD übertragene Passwort möglicherweise nicht mehr den Anforderungen entspricht. Ansonsten bedeutet der Wert -3, dass das mit dem Tool KDCUPD übertragene Passwort tatsächlich nicht den Anforderungen der generierten Komplexitätsstufe entspricht oder zu kurz ist. Das Passwort muss mit SIGN CP geändert werden, wenn der Anmelde-Vorgang erfolgreich beendet werden soll.
Dieser Wert ist nur möglich, wenn Grace-Sign-On erlaubt ist (SIGNON-Anweisung der Generierung, Parameter GRACE = YES).

KCRPWMIN

Bei erfolgreicher Anmeldung (KCRSIGN1 = A/R) oder noch nicht erfolgreicher Anmeldung (KCRSIGN1 = I) enthält das Feld die Anzahl Tage, die das Passwort dieses Benutzers nicht durch einen SIGN CP Aufruf geändert werden darf.
Der Wert 0 bedeutet, dass das Passwort geändert werden darf.

KCRUSER

Nach nicht erfolgreicher oder noch nicht vollständiger Anmeldung (KCRSIGN1 = U/I) enthält das Feld den Namen des Benutzers, der abgelehnt wurde (KCRSIGN1 = U) oder für den noch ein Zwischendialog durchgeführt werden muss (KCRSIGN1 = I), ansonsten Leerzeichen.

KCRTAC

Im Anmelde-Vorgang für den UPIC-Client enthält das Feld den Namen des im UPIC-Protokoll übergebenen Transaktionscode (TP_Name), ansonsten Leerzeichen. Es wurde von openUTM nicht geprüft, ob es sich um einen gültigen Transaktionscode handelt.

KCLSTSGN

Nach erfolgreicher Anmeldung (KCRSIGN1 = A/R) oder noch nicht erfolgreicher Anmeldung (KCRSIGN1 = I) eines Benutzers enthält das Feld das Datum und die Uhrzeit des letzten erfolgreichen Anmeldens dieses Benutzers an die Anwendung. Datum und Uhrzeit werden in der Form YYYYMMDDHHMMSS übergeben. Nach dem ersten erfolgreichen Anmelden nach einer Neu-Generierung werden abdruckbare Nullen zurückgegeben.

KCDSPMSG

Nach erfolgreicher Anmeldung (KCRSIGN1 = A/R) eines Benutzers enthält das Feld folgenden Wert:

Y

wenn für den Benutzer ein offener Dialog-Vorgang (KCRSIGN1=R) vorliegt oder eine Dialog-Nachricht, die mit MPUT PM ausgegeben werden kann.

N

in allen anderen Fällen.

KCTAPTC

Das Feld enthält nach erfolgreicher Anmeldung eines Benutzers mit nachfolgendem Vorgangswiederanlauf (KCRSIGN1 = R) folgenden Wert:

Y

wenn für den Benutzer eine Transaktion im Zustand P(repare) T(o) C(ommit) existiert. In diesem Fall darf der Vorgangs-Wiederanlauf nicht unterdrückt werden.

N

in allen anderen Fällen.

KCCLNODE

Unix-, Linux- und Windows-Systeme
Das Feld enthält bei nicht erfolgreicher Anmeldung (KCRSIGN1 = U) wegen eines an eine andere Knoten-Anwendung gebundenen Vorgangs (KCRSIGN2 = 25) den Rechnernamen des Knotens, an den der offene Vorgang gebunden ist.

KCRPASSL

Im Anmelde-Vorgang für den UPIC-Client enthält das Feld nach erfolgreicher Anmeldung (KCRSIGN1 = A /R) eines Benutzers, der ohne Passwort generiert ist, das im UPIC-Protokoll übergebene Passwort, ansonsten Leerzeichen.

- Angaben im Nachrichtenbereich bei SIGN ON und SIGN CP
  - Bei SIGN ON ist das Passwort in der Länge von 16 Bytes in den Nachrichtenbereich zu schreiben. Leerzeichen bedeuten "Benutzerkennung ohne Passwort".
  - Bei SIGN CP sind altes und neues Passwort in der Länge von je 16 Bytes wie folgt in den Nachrichtenbereich zu schreiben:

Passwort alt[1]	Passwort neu[1]

    [1]Leerzeichen bedeuten jeweils "Benutzerkennung ohne Passwort"

    Das neue Passwort muss aus Zeichen bestehen, die in der UTM-Partner-Anwendung erlaubt sind, siehe openUTM-Handbuch „Anwendungen generieren“, USER-Anweisung.

    Ist der Aufruf syntaktisch korrekt, dann überschreibt openUTM den Datenbereich mit Leerzeichen.

    Die Administrationsberechtigung ist für diesen Aufruf nicht erforderlich.

- Bei SIGN ON prüft openUTM, ob eine Anmeldung des Benutzers zu diesem Zeitpunkt von diesem Client aus möglich ist.
- Bei SIGN CK prüft openUTM, ob die Berechtigungsdaten für eine erfolgreiche Anmeldung von diesem Client aus ausreichen, nicht aber, ob eine Anmeldung zu diesem Zeitpunkt möglich ist.
- SIGN OF und SIGN OB dürfen nur in solchen Teilprogrammen gegeben werden, die mit PEND RE oder PEND FI beendet werden und die eine Dialog-Nachricht an das Terminal, das UPIC-Client oder die Transportsystem-Anwendung ausgeben, andernfalls bricht openUTM den Vorgang mit PEND ER ab.

  SIGN OB an UPIC-Clients oder Transportsystem-Anwendungen wirkt wie SIGN OF.SIGN OF und SIGN OB wirken erst bei der nächsten Eingabe vom Terminal, d.h. der Benutzer wird erst nach der nächsten Eingabe abgemeldet (SIGN OB) bzw. die Verbindung zum Terminal wird erst nach der nächsten Eingabe abgebaut (SIGN OF). Bei UPIC-Clients oder Transportsystem-Anwendungen wird der Verbindungsabbau sofort initiiert.
- SIGN ST und SIGN ON sind nur im Anmelde-Vorgang erlaubt.

> ¡ Der symbolische Name, der in der COBOL-Datenstruktur für den Operationscode SIGN verwendet wird, lautet SGN, da SIGN ein reserviertes COBOL-Wort ist.

7.25.1 SIGN CL - Locale der Benutzerkennung ändern (BS2000-Systeme)

Mit SIGN CL (Change Local) kann das Benutzer-spezifische Locale für die aktuelle Benutzerkennung geändert werden, d.h. das Sprachkennzeichen, das Territorialkennzeichen und der Namen des verwendeten Zeichensatzes können neu gesetzt werden, wenn es sich bei der aktuellen Benutzerkennung um eine echte Benutzerkennung handelt (keine Verbindungs-Benutzerkennung).

SIGN CL ist nur in Dialog-Teilprogrammen einer UTM-Anwendung erlaubt, die mit Benutzerkennungen konfiguriert ist. Administrationsberechtigung ist nicht erforderlich.

Ist der Aufruf erfolgreich, so gilt vom nächsten Transaktionsende an das neue Locale für diese Benutzerkennung. Alle Nachrichten innerhalb der aktuellen Transaktion werden noch unter Verwendung des alten Zeichensatznamens aufbereitet. Deshalb ist es sinnvoll, das Teilprogramm mit Transaktionsende, aber ohne Ausgabe einer Dialog-Nachricht an das Terminal zu beenden, wenn der Zeichensatzname des Benutzers geändert wurde.

Wollen Sie nur bestimmte Komponenten des Locale ändern, so müssen Sie die restlichen Komponenten mit binär null versorgen.

Änderungen, die mit SIGN CL vorgenommen wurden, bleiben auch nach einem KDCUPD-Lauf erhalten. KDCUPD überträgt implizit für jeden Benutzer die aktuellen Werte seines Locale in die neue KDCFILE.

SIGN CL ist eine aufwärtskompatible Erweiterung gegenüber DIN 66265.

Versorgung des KDCS-Parameterbereichs (1. Parameter) bei SIGN CL

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich				
	KCOP	KCOM	KCLANGID	KCTERRID	KCCSNAME
Das Locale der Benutzerkennung ändern	SIGN	CL	neues Sprachkennzeichen der Benutzerkennung	neues Territorialkennzeichen der Benutzerkennung	CCS-Name des neuen Zeichensatzes für die Benutzerkennung

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"SIGN"
KCOM	"CL"
KCLANGID	Sprachkennzeichen des Benutzers / binär null
KCTERRID	Territorialkennzeichen des Benutzers / binär null
KCCSNAME	Zeichensatzname des Benutzers / binär null

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++Makroaufruf	
Makronamen	Parameter
KDCS_SIGNCL	(nb,kclangid,kcterrid,kcccsname)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

Im KDCS-Parameterbereich machen Sie für den SIGN-Aufruf mit Operationsmodifikation CL folgende Einträge:

KCOP

im Feld KCOP den Operationsnamen SIGN.

KCOM

im Feld KCOM die Operationsmodifikation

- CL (Change Locale): Locale der Benutzerkennung ändern.

KCLANGID

im Feld KCLANGID das neue Sprachkennzeichen, das der Benutzerkennung, von der der Vorgang gestartet wurde, zugeordnet werden soll. Die Länge des Sprachkennzeichens ist 2 Byte. Soll das Sprachkennzeichen nicht geändert werden, ist für KCLANGID binär null anzugeben.

KCTERRID

im Feld KCTERRID das neue Territorialkennzeichen, das der Benutzerkennung, von der der Vorgang gestartet wurde, zugeordnet werden soll. Die Länge des Territorialkennzeichens ist 2 Byte. Soll das Territorialkennzeichen nicht geändert werden, ist für KCTERRID binär null anzugeben.

KCCSNAME

im Feld KCCSNAME der CCS-Name des neuen Zeichensatzes, der der Benutzerkennung zugeordnet werden soll. Die Länge des CCS-Namens ist maximal 8 Byte. Soll kein neuer Zeichensatz zugeordnet werden, ist für KCCSNAME binär null anzugeben.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   als 1. Parameter: die Adresse des KDCS-Parameterbereichs

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode (Länge 3 Byte). Die möglichen Returncodes und ihre Bedeutung siehe unten.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim SIGN CL-Aufruf

Im Programm sind auswertbar:

000 Die Operation wurde ausgeführt, die angegebene(n) Komponente(n) des Locale geändert.

40Z Das System konnte die Operation nicht ausführen (Generierungsfehler).

41Z Der Aufruf SIGN CL ist an dieser Stelle nicht erlaubt:

- Der Aufruf erfolgte vor erfolgreicher Anmeldung.
- Der Aufruf erfolgte in einer Anwendung ohne Benutzerkennung.

46Z Die Angaben zum neuen Locale sind falsch. Das Locale der Benutzerkennung wird nicht geändert. Die genauere Ursache liefert openUTM im Feld KCRCDC.

49Z Felder des KDCS-Parameterbereichs, die von diesem Aufruf nicht verwendet werden, sind nicht mit binär null gelöscht.

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In dem Teilprogrammlauf wurde noch kein INIT-Aufruf gegeben.

7.26 SPUT Schreiben in einen Sekundärspeicherbereich

Mit einem Aufruf SPUT (storage PUT) schreiben Sie Daten aus einem angegebenen Bereich in einen

- Globalen Sekundären Speicherbereich (GSSB) oder in einen
- Lokalen Sekundären Speicherbereich (LSSB) oder in einen
- User-spezifischen Langzeitspeicher (ULS).

Beachten Sie, dass der Name eines ULS-Blocks bei der Generierung vergeben wird (ULS-Anweisung bei KDCDEF), während Sie die Namen von GSSBs und LSSBs beim SPUT-Aufruf frei wählen können.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen stehen GSSB oder ULS Cluster-weit zur Verfügung.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**				
	KCOP	**KCOM**	**KCLA**	**KCRN**	**KCUS**
Schreiben in einen LSSB	"SPUT"	"DL"/ "MS"/"ES" (jeweils gleiche Wirkung)	Länge	Name des LSSB	-
Schreiben in einen GSSB	"SPUT"	"GB"	Länge	Name des GSSB	-
Schreiben in einen ULS	"SPUT"	"US"	Länge	Blockname	Benutzerkennung/LSES-Name /Association-Name/Leerzeichen

Bei KCOM = US müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen des 2. Parameters

Hier stellen Sie die Adresse des Nachrichtenbereichs bereit, der die zu schreibende Nachricht enthält.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"SPUT"
KCOM	"GB"/"DL"/"MS"/"ES"/"US" ("DL", "MS" und "ES" haben jeweils die gleiche Wirkung)
KCLA	Länge in Byte
KCRN	Name des Bereichs
KCUS	Benutzerkennung/LSES-Name/Association-Name/Leerzeichen/-
Nachrichtenbereich	
	Daten

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_SPUTGB/KDCS_SPUTDL/KDCS_SPUTMS/KDCS_SPUTES	(nb,kcla,kcrn)
KDCS_SPUTUS	(nb,kcla,kcrn,kcus)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den SPUT-Aufruf ein:

KCOP

im Feld KCOP den Operationscode SPUT.

KCOM

im Feld KCOM die Angabe,

- ob in einen LSSB geschrieben werden soll ("DL" oder "MS" oder "ES") oder
- ob in einen GSSB geschrieben werden soll ("GB") oder
- ob in einen ULS-Block geschrieben werden soll ("US").

Die Einträge "MS" und "ES" haben bei openUTM die gleiche Wirkung wie "DL".

KCLA

im Feld KCLA die Länge der Daten, die Sie im Nachrichtenbereich zur Übergabe bereitstellen. Die Länge wird nicht in den LSSB/GSSB/ULS geschrieben.

KCRN

im Feld KCRN den Namen des LSSB/GSSB oder ULS-Blocks, der angelegt werden soll bzw. in den die Daten geschrieben werden sollen. Leerzeichen und binär null sind ungültige Einträge.

KCUS

im Feld KCUS die Benutzerkennung/LSES-Name/Association-Name (bei KCOM=US), wenn ein ULS-Block einer fremden Benutzerkennung/Session/Association beschrieben werden soll, sonst Leerzeichen. Wird eine fremde Benutzerkennung/LSES-Name/Association-Name in KCUS eingetragen, dann muss die eigene Benutzerkennung administrationsberechtigt sein.

Bei KCOM = DL/MS/ES/GB: irrelevant.

Nachrichtenbereich

Im Nachrichtenbereich tragen Sie die Nachricht ein, die Sie ausgeben wollen.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

2. Parameter

als 2. Parameter: die Adresse des Nachrichtenbereichs, aus dem openUTM die Nachricht lesen soll. Die Adresse des Nachrichtenbereichs müssen Sie auch dann angeben, wenn Sie in KCLA die Länge 0 eintragen.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode, siehe nächste Seite.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim SPUT-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout), siehe KCRCDC.
41Z	Der Aufruf wurde im ersten Teil des Anmelde-Vorgangs gegeben, obwohl dies in der Generierung nicht erlaubt wurde. bei KCOM=US: der Aufruf wurde im ersten Teil des Anmelde-Vorgangs oder nach einem SIGN ON und vor dem PEND PS Aufruf abgesetzt.
42Z	Der Eintrag in KCOM ist ungültig.
43Z	Die Längenangabe in KCLA ist negativ bzw. ungültig.
44Z	Der Name in KCRN ist ungültig. Er ist ungültig, wenn er nur aus Leerzeichen oder nur aus binär null besteht oder nicht generiert ist (bei ULS).
46Z	Die Angabe in KCUS ist ungültig.
47Z	Der Nachrichtenbereich fehlt oder ist in der angegebenen Länge nicht zugreifbar.
49Z	Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null (nur bei KCOM = US).

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z	In diesem Programm wurde kein INIT gegeben.

Die folgende Tabelle beschreibt, wie die Aufruffolge SPUT . . . RSET oder SPUT . . . PEND/PGWT auf GSSBs, ULSs und LSSBs wirkt

Aufruf		Wirkung auf GSSBs/ULSs	Wirkung auf LSSBs
SPUT		sperrt; falls noch nicht vorhanden, werden GSSBs erzeugt, vorhandene GSSBs werden ersetzt	erzeugt oder ersetzt
...	PEND KP PGWT KP/PR	lässt rücksetzbar und gesperrt	lässt rücksetzbar
...	PEND RE /SP PGWT CM	setzt gültig (sie sind nicht mehr rücksetzbar) und entsperrt (d.h. andere Transaktionen können sie verwenden)	setzt gültig; sie sind nicht mehr rücksetzbar
....	PEND FI /FC		löscht; sie sind nicht mehr verfügbar
....	RSET PEND RS PGWT RB	macht Änderungen rückgängig und entsperrt; ein GSSB wird gelöscht, wenn er in dieser Transaktion erzeugt wurde	macht Änderungen rückgängig
....	PEND ER /FR		löscht

Beachten Sie bitte folgende Eigenschaften von GSSBs, ULSs und LSSBs:

- Ein GSSB steht allen Teilprogrammen einer Anwendung zur Verfügung, d.h. er kann von allen Teilprogrammen überschrieben werden. Sie müssen durch entsprechende Namensvergabe dafür sorgen, dass er von anderen Teilprogrammen nicht unbeabsichtigt überschrieben werden kann.
- *Unix-, Linux- und Windows-Systeme*
  In UTM-Cluster-Anwendungen stehen GSSB und ULS Cluster-weit zur Verfügung. D.h. ein GSSB oder ULS, den Sie mit SPUT erzeugen/schreiben, existiert in jeder Knoten-Anwendungen und kann dort mit SGET gelesen werden.
- Ein LSSB ist einem Vorgang eindeutig zugeordnet.
- Ein GSSB, LSSB oder ULS-Block hat immer die Länge des zuletzt aufgerufenen SPUT. Sie können maximal 32767 Bytes lang sein.
- Der Name eines ULS-Blocks wird bei der Generierung definiert (wie ein TLS-Block).
- Die maximale Anzahl der GSSBs bzw. LSSBs wird bei der Generierung festgelegt.

Eigenschaften des SPUT-Aufrufs

- Der SPUT-Aufruf für einen GSSB/ULS sperrt diesen GSSB bzw. ULS-Block bis zum Ende der Transaktion d.h. bis PEND RE, SP, FI, FC, RS oder ER/FR.

  Ein GSSB/ULS-Block bleibt nach einem SPUT-Aufruf bei einem nachfolgenden PEND KP bzw. PGWT KP-Aufruf oder einem PEND PA/PR bzw. PGWT PR (beim Warten auf eine DGET-Nachricht) über das Ende des Verarbeitungs-/Dialog-Schrittes hinweg von dieser Transaktion gesperrt.
  Eine andere Transaktion, die diesen GSSB/ULS-Block mit SGET, SPUT oder SREL bearbeiten will, wird zurückgewiesen bei:

  - PEND KP und PGWT KP
  - PEND PA/PR mit Taskwechsel wegen TAC-Klassensteuerung
  - PEND PA/PR oder PGWT PR mit Warten auf eine DGET-Nachricht.

  Bei Transaktionsende oder Rücksetzoperationen werden alle gesperrten GSSBs und ULS-Blöcke freigegeben.

  Wie openUTM reagiert, wenn der gewünschte GSSB oder ULS-Block gesperrt ist, ist in Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“ beschrieben.
- GSSBs mit Länge 0 in KCLA sind zulässig. Über einen solchen GSSB können sich Anwendungsprogramme verständigen, wobei nur ausgewertet wird, ob der GSSB gesperrt ist oder nicht. Einen GSSB mit der Länge 0 löscht openUTM jedoch beim nächsten Start der Anwendung. Auch KDCUPD überträgt GSSBs der Länge 0 nicht in eine neue KDCFILE (siehe openUTM-Handbuch „Anwendungen generieren“, KDCFILE ändern).
- Ein SPUT-Aufruf mit KCLA=0 kann auch dazu verwendet werden, um den Inhalt von ULS-Blöcken zu löschen.

7.27 SREL Löschen eines Sekundärspeicherbereichs

Mit dem Aufruf SREL (storage release) löschen Sie einen Sekundärspeicherbereich. Als Sekundärspeicherbereiche kommen dabei infrage:

- der Globale Sekundäre Speicherbereich (GSSB)
- der Lokale Sekundäre Speicherbereich (LSSB)

Blöcke eines ULS (User-spezifischer Langzeitspeicher) können **nicht** mit SREL gelöscht werden, da ihre Namen bei der Generierung der Anwendung vergeben werden. Soll der Inhalt eines ULS-Block gelöscht werden, so ist der Block mit der Länge null zu überschreiben.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen ist ein GSSB Cluster-weit gültig. Damit wirkt sich auch sein Löschen mit SREL Cluster-weit aus.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	Einträge im KDCS-Parameterbereich		
	KCOP	KCOM	KCRN
Löschen eines LSSB	"SREL"	"LB"	Name des LSSB
Löschen eines GSSB	"SREL"	"GB"	Name des GSSB

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"SREL"
KCOM	"LB"/"GB"
KCRN	Name des Bereichs

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	—

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_SRELGB / KDCS_SRELLB	(kcrn)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den SREL-Aufruf ein:

KCOP

im Feld KCOP den Operationscode SREL.

KCOM

im Feld KCOM

- LB zum Löschen eines LSSBs oder
- GB zum Löschen eines GSSBs.

KCRN

im Feld KCRN den Namen des LSSB/GSSB, der gelöscht werden soll.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim SREL-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
14Z	Unter dem in KCRN angegebenen Namen ist kein Bereich vorhanden.
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout), siehe KCRCDC.

42Z Der Eintrag in KCOM ist ungültig.

44Z Name in KCRN ist ungültig (wenn er nur aus Leerzeichen oder nur aus binär null besteht).

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Eigenschaften des SREL-Aufrufs

- openUTM führt SREL erst am Ende der laufenden Transaktion aus.
- SREL wird nicht ausgeführt
  - bei Vorgangs-Abbruch durch PEND ER/FR, oder
  - wenn ein PEND RS oder ein RSET-Aufruf folgt.
- Ein SREL-Aufruf sperrt den aufgerufenen Bereich bis zum Transaktionsende bzw. der Freigabe durch UNLK oder bis zur nächsten Rücksetzoperation, d.h. nachfolgende SGET-Aufrufe auf diesen Bereich weist openUTM mit dem Returncode 14Z zurück. Folgt aber auf einen SREL-Aufruf ein SPUT-Aufruf mit demselben Bereichsnamen, dann wird dieser Bereich (LSSB oder GSSB) neu eingerichtet.

  Die Sperre wird über einen Verarbeitungs-/Dialog-Schritt hinaus beibehalten, wenn der Teilprogrammlauf abgeschlossen oder unterbrochen wird mit
  - PEND KP und PGWT KP,
  - PEND PA/PR mit Taskwechsel wegen TAC-Klassensteuerung,
  - PEND PA/PR oder PGWT PR mit Warten auf eine DGET-Nachricht.

  Ist der gesperrte Bereich ein GSSB, dann können andere Vorgänge auf diesen GSSB weder mit SGET noch mit SPUT zugreifen. Wie openUTM in diesem Fall reagiert, ist in Abschnitt „Verhalten bei gesperrten Speicherbereichen (TLS, ULS und GSSB)“ beschrieben.
- Bei Vorgangsende (PEND FI/FC) oder Vorgangs-Abbruch (PEND ER/FR, u.U. auch bei PEND RS) löscht openUTM automatisch alle LSSBs. SREL-Aufrufe auf LSSBs sind daher in Transaktionen, die den Vorgang beenden, überflüssig.
- *Unix-, Linux- und Windows-Systeme*
  In UTM-Cluster-Anwendungen wird ein GSSB durch einen SREL-Aufruf Cluster-weit gelöscht, d.h. sobald der SREL-Aufruf wirksam wird, kann der GSSB in keiner Knoten-Anwendung mehr gelesen werden.

Die folgende Tabelle beschreibt, wie die Aufruffolge SREL . . . RSET oder SREL . . . PEND auf GSSBs oder LSSBs wirkt.

Aufruf		**Wirkung auf GSSB**	**Wirkung auf LSSB**
SREL		**löscht und sperrt; Operation bleibt rücksetzbar**	**löscht; Operation bleibt rücksetzbar**
...	RSET PEND RS PGWT RB	setzt zurück und entsperrt	setzt zurück
...	PEND KP PGWT KP	lässt die Operation rücksetzbar	lässt die Operation rücksetzbar
....	PEND RE PEND SP PGWT CM	löscht und entsperrt; Operation nicht rücksetzbar	löscht; Operation nicht rücksetzbar
....	PEND FI PEND FC		löscht alle LSSBs
....	PEND ER PEND FR	setzt zurück und entsperrt	löscht alle LSSBs
....	PEND PA/PR [1] PGWT PR [1]	Sperre bleibt, Operation rücksetzbar	Sperre bleibt, Operation rücksetzbar

[1] Mit Taskwechsel wegen TAC-Klassensteuerung oder mit Warten auf eine DGET-Nachricht

7.28 UNLK Entsperren eines TLS, ULS oder eines GSSB

Mit dem Aufruf UNLK (unlock) entsperren Sie einen der folgenden Speicherbereiche:

- den Globalen Sekundären Speicherbereich (GSSB).
- einen Block des Terminal-spezifischen Langzeitspeichers (TLS).
- einen Block des User-spezifischen Langzeitspeichers (ULS).

Der Bereich wird nur dann entsperrt, wenn aus ihm in der aktuellen Transaktion nur gelesen wurde.

Unix-, Linux- und Windows-Systeme
In UTM-Cluster-Anwendungen sind GSSB und ULS Cluster-weit gültig. Damit wirkt sich das Entsperren eines GSSB oder ULS mit UNLK Cluster-weit aus.

Versorgen des 1. Parameters (KDCS-Parameterbereich)

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten und die dafür notwendigen Angaben im KDCS-Parameterbereich.

Funktion des Aufrufs	**Einträge im KDCS-Parameterbereich**			
	KCOP	**KCOM**	**KCRN**	**KCLT bzw. KCUS**
Entsperren eines TLS (in Dialog-Programmen)	"UNLK"	"DA"	Blockname	-
Entsperren eines TLS (in Asynchron-Programmen)	"UNLK"	"DA"	Blockname	LTERM/LPAP/OSI-LPAP/Master-LPAP-Name
Entsperren eines GSSB	"UNLK"	"GB"	Name des GSSB	-
Entsperren eines ULS	"UNLK"	"US"	Blockname	Benutzerkennung/LSES-Name /Association-Name/Leerzeichen

Bei KCOM = US müssen alle nicht verwendeten Felder des KDCS-Parameterbereichs mit binär null versorgt werden.

Versorgen der Parameter	
Feldname im KDCS-Parameterbereich	Inhalt
KCOP	"UNLK"
KCOM	"GB"/"DA"/"US"
KCRN	Name des Bereichs/Blockname
KCLT bzw. KCUS	LTERM/LPAP/OSI-LPAP/Master-LPAP-Name bzw. Benutzerkennung/LSES-Name/Association-Name/Leerzeichen

KDCS-Aufruf	
1. Parameter	2. Parameter
KDCS-Parameterbereich	Nachrichtenbereich

C/C++-Makroaufrufe	
Makronamen	Parameter
KDCS_UNLKGB	(kcrn)
KDCS_UNLKDA	(kcrn,kclt)
KDCS_UNLKUS	(kcrn,kcus)

Rückgaben von openUTM	
Feldname im KB-Rückgabebereich	**Inhalt**
KCRCCC	Returncode
KCRCDC	interner Returncode

In den KDCS-Parameterbereich tragen Sie für den UNLK-Aufruf ein:

KCOP

im Feld KCOP den Operationscode UNLK.

KCOM

im Feld KCOM den Speichertyp, der entsperrt werden soll:

- GB für einen Globalen Sekundären Speicherbereich (GSSB),
- DA für einen Terminal-spezifischen Langzeitspeicher (TLS),
- US für einen User-spezifischen Langzeitspeicher (ULS).

KCRN

im Feld KCRN den Namen des zu entsperrenden Speicherbereichs.

KCLT
KCUS

je nachdem, um welchen Speichertyp es sich handelt:

- beim Entsperren eines TLS in einem Asynchron-Programm:
  im Feld **KCLT** den Namen des LTERM- oder (OSI-)LPAP-Partners, dessen TLS entsperrt werden soll,
- beim Entsperren eines TLS in einem Dialog-Programm:
  irrelevant, es wird immer auf den entsprechenden Block des "eigenen" TLS zugegriffen.

- beim Entsperren eines ULS-Blocks:
  im Feld **KCUS** die Benutzerkennung, wenn ein ULS-Block einer fremden Benutzerkennung entsperrt werden soll, oder Leerzeichen bei einem ULS-Block der eigenen Benutzerkennung. Wird eine fremde Benutzerkennung in KCUS eingetragen, dann muss die eigene Benutzerkennung administrationsberechtigt sein.

  Falls ein ULS-Block einer fremden Session/Association entsperrt werden soll, ist deren Name anzugeben.
- beim Entsperren eines GSSBs:
  irrelevant.

Beim KDCS-Aufruf geben Sie an:

1. Parameter

   als 1. Parameter: Die Adresse des KDCS-Parameterbereichs.

Makronamen

Wie Sie Makroaufrufe für C/C++ nutzen, ist in Abschnitt „C/C++-Makroschnittstelle“ ausführlich beschrieben.

openUTM gibt zurück:

KCRCCC

im Feld KCRCCC den KDCS-Returncode.

KCRCDC

im Feld KCRCDC den internen Returncode von openUTM (siehe openUTM-Handbuch „Meldungen, Test und Diagnose“).

KDCS-Returncodes im Feld KCRCCC beim UNLK-Aufruf

Im Programm sind auswertbar:

000	Die Operation wurde ausgeführt.
14Z	Unter dem in KCRN angegebenen Namen ist kein GSSB/TLS vorhanden.
16Z	GSSB/TLS/ULS nicht von eigener Transaktion gesperrt oder GSSB wurde in gleicher Transaktion erzeugt oder geändert oder TLS wurde in gleicher Transaktion geändert.
40Z	Das System kann die Operation nicht ausführen (Generierungs- bzw. Systemfehler, Deadlock, Timeout).
42Z	Die Angabe in KCOM ist ungültig.
44Z	Bei GSSB: Angabe in KCRN ist ungültig (Leerzeichen oder binär null). Bei TLS und ULS: Der Name des Blocks in KCRN ist unbekannt oder ungültig.
46Z	LTERM- oder LPAP-Name in KCLT ist ungültig (nur bei Asynchron-Programmen und TLS) oder die Benutzerkennung ist unbekannt (nur bei ULS).
49Z	Der Inhalt nicht verwendeter Felder des KDCS-Parameterbereichs ist ungleich binär null (nur bei KCOM = US).

Ein weiterer Returncode ist dem DUMP zu entnehmen:

71Z In diesem Programm wurde kein INIT gegeben.

Eigenschaften des UNLK-Aufrufs

- Ein UNLK-Aufruf auf einen TLS/ULS/GSSB ist nur sinnvoll nach einem vorherigen Leseaufruf (GTDA bzw. SGET). Wurde der Bereich in der aktuellen Transaktion mit PTDA, SPUT oder SREL (bei GSSBs) verändert, dann weist openUTM den Aufruf mit KCRCCC = 16Z zurück; der Bereich bleibt bis Transaktionsende gesperrt.
- Ein UNLK-Aufruf ist z.B. vor einem PEND KP bzw. PGWT KP/PR sinnvoll. Dadurch lassen sich TLS/ULS-Blöcke und GSSBs bereits vor dem nächsten Sicherungspunkt freigeben. Damit stehen sie den anderen Transaktionen zur Verfügung.
- Ein UNLK-Aufruf für einen Bereich, der nicht von der eigenen Transaktion gesperrt ist, wird abgewiesen (Returncode 16Z).
- *Unix-, Linux- und Windows-Systeme*
  In UTM-Cluster-Anwendungen wird ein GSSB oder ULS durch einen UNLK-Aufruf Cluster-weit entsperrt, d.h. sobald der UNLK-Aufruf wirksam wird, können auch Transaktionen in anderen Knoten-Anwendungen wieder auf den entsperrten Bereich zugreifen.

8 Programmschnittstelle UTM-HTTP

In diesem Kapitel finden Sie alle Informationen, die Sie zum Einsatz der Programmschnittstelle UTM-HTTP in Ihrem Programm benötigen. Mit der Programmschnittstelle UTM-HTTP stellt Ihnen openUTM C-Funktionen zur Verfügung, die Sie in ihren Teilprogrammen und im „Event-Exit HTTP" verwenden können.

Zusätzlich finden Sie in diesem Kapitel Detailinformationen zur Kommunikation mit HTTP-Clients.

Einen Überblick über die Kommunikation mit HTTP-Clients finden Sie in den Kapiteln "Aus dem Internet auf UTM-Anwendungen zugreifen" und "Kommunikation mit HTTP-Clients" im openUTM-Handbuch „Konzepte und Funktionen“.

8.1 Übersicht über alle UTM-HTTP-Funktionen

Die Funktionen der UTM-HTTP-Programmschnittstelle teilen sich in zwei Gruppen auf, die `kcHttpGet`- und die `kcHttpPut`-Funktionen. Sie können sowohl von einem HTTP-Exit-Programm (s. Kapitel „Event-Exit HTTP") als auch von UTM-Teilprogrammen aufgerufen werden. Eine Ausnahme bilden die Funktionen `kcHttpGetMputMsg`, `kcHttpPutMgetMsg`, `kcHttpPutRspMsgBody`, die nur in einem HTTP-Exit-Programm verwendet werden dürfen. Außerdem gibt es die beiden Hilfsfunktionen `kcHttpPercentDecode` und `kcHttpGetRc2String`.

In UTM-Teilprogrammen dürfen die Funktionen erst nach dem KDCS-Aufruf INIT aufgerufen werden. Die im Teilprogramm erlaubten `kcHttpGet`-Funktionen dürfen nur im ersten Teilprogrammlauf eines (nicht geketteten) Vorgangs aufgerufen werden.
Wenn ein Teilprogramm die Funktionen `kcHttpPutHeader` und `kcHttpPutStatus` verwendet, so muss es auch mit MPUT eine Nachricht für den HTTP-Client bereitstellen und den Vorgang beenden, ansonsten gehen die übergebenen Header- bzw. Status-Informationen verloren.
Zur Verwendung der Funktionen im HTTP-Exit-Programm s. Kapitel „Event-Exit HTTP".

Eingabe-Parameter der C-Funktionen, die vom Typ `char *` sind, müssen vom Aufrufer als null-terminierter String übergeben werden. Ebenso gibt openUTM in Ausgabe-Parametern, die vom Typ `char *` sind, null-terminierte Strings zurück.

Alle Funktionen bis auf die Hilfsfunktion `kcHttpGetRc2String` liefern bei nicht erfolgreicher Ausführung negative Zahlenkonstanten, die als C-Datentyp `enum` definiert sind, als Funktionsergebnis zurück.
Bei erfolgreicher Ausführung wird entweder die Zahlenkonstante 0 oder ein positiver Funktionswert zurückgeliefert.

Folgende Funktionen stehen zur Verfügung:

Funktion	Beschreibung
`kcHttpGetHeaderByIndex`	liefert den Namen und Wert des HTTP-Header-Feldes für den angegebenen Index zurück
`kcHttpGetHeaderByName`	liefert den Wert des über den Namen spezifizierten HTTP-Header-Feldes zurück
`kcHttpGetHeaderCount`	liefert die Anzahl der im HTTP-Request enthaltenen HTTP-Header-Felder zurück
`kcHttpGetMethod`	liefert die HTTP-Methode des HTTP-Requests zurück
`kcHttpGetMputMsg`	liefert die vom Teilprogramm erzeugte MPUT-Nachricht zurück
`kcHttpGetPath`	liefert den mit normierten Path des HTTP-Requests zurück
`kcHttpGetQuery`	liefert die mit normierte Query des HTTP-Requests zurück
`kcHttpGetRc2String`	Hilfsfunktion zur Umwandlung eines Funktionsergebnisses vom Typ `enum` in einen abdruckbaren null-terminierten String
`kcHttpGetReqMsgBody`	liefert den Message Body des HTTP-Requests zurück
`kcHttpGetScheme`	liefert das Scheme des HTTP-Requests zurück
`kcHttpGetVersion`	liefert die HTTP-Version des HTTP-Requests zurück

Funktion	Beschreibung
kcHttpPercentDecode	wandelt Zeichen in Prozent-Darstellung in ihre normale Ein-Zeichen-Darstellung um
kcHttpPutHeader	übergibt ein HTTP-Header-Feld für die HTTP-Response
kcHttpPutMgetMsg	übergibt eine Nachricht für das Teilprogramm, die mit MGET gelesen werden kann
kcHttpPutRspMsgBody	übergibt eine Nachricht für den Message Body der HTTP-Response
kcHttpPutStatus	übergibt einen HTTP-Statuscode für die HTTP-Response

Folgende Returncodes sind definiert:

Returncodes

```
typedef enum
   {  KC_HTTP_OK                                    =    0
    , KC_HTTP_RESULT_TRUNCATED                      =   -1
    , KC_HTTP_FUNCTION_CALL_NOT_ALLOWED             =   -2
    , KC_HTTP_NO_HTTP_CLIENT                        =   -3
    , KC_HTTP_INVALID_INDEX                         =   -4
    , KC_HTTP_HEADER_NAME_NULL_OR_EMPTY             =   -5
    , KC_HTTP_HEADER_VALUE_NULL_OR_EMPTY            =   -6
    , KC_HTTP_HEADER_NAME_TOO_LONG                  =   -7
    , KC_HTTP_HEADER_VALUE_TOO_LONG                 =   -8
    , KC_HTTP_PARAM_VALUE_NULL                      =   -9
    , KC_HTTP_INVALID_LENGTH                        =  -10
    , KC_HTTP_INVALID_STATUS_CODE                   =  -11
    , KC_HTTP_STATUS_CODE_NOT_ALLOWED               =  -12
    , KC_HTTP_HEADER_NOT_FOUND                      =  -13
    , KC_HTTP_HEADER_NOT_ALLOWED                    =  -14
    , KC_HTTP_REASON_PHRASE_TOO_LONG                =  -15
    , KC_HTTP_MEMORY_INSUFFICIENT                   =  -16
    , KC_HTTP_NORMALIZATION_ERROR                   =  -17
    , KC_HTTP_MAX_HEADER_COUNT                      =  -18
    , KC_HTTP_MAX_HEADER_TOTAL_LENGTH               =  -19
    , KC_HTTP_HEADER_NAME_NOT_PRINTABLE             =  -20
    , KC_HTTP_HEADER_VALUE_NOT_PRINTABLE            =  -21
    , KC_HTTP_REASON_PHRASE_NOT_PRINTABLE           =  -22
    , KC_HTTP_PARAM_INVALID_URLSTRING               =  -23
   } kc_http_retcode;
```

Bei der Beschreibung der einzelnen Funktionen werden die Funktionsparameter wie folgt gekennzeichnet:

>>	Eingabeparameter	Eingabewert oder Adresse, die beim Funktionsaufruf einen Eingabewert enthält.
<<	Ausgabeparameter	Adresse, die beim Funktionsaufruf einen beliebigen Wert und nach Rückkehr von der Funktion einen Ausgabewert enthält.
<>	Ein-/Ausgabeparameter	Adresse, die beim Funktionsaufruf einen Eingabewert und nach Rückkehr von der Funktion einen Ausgabewert enthält.

In den folgenden Kapiteln werden die einzelnen Funktionen detailliert beschrieben.

8.1.1 Funktion kcHttpGetHeaderByIndex

Die Funktion `kcHttpGetHeaderByIndex` liefert den Namen und Wert des HTTP-Header-Feldes für den angegebenen Index zurück.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetHeaderByIndex

```
kc_http_retcode kcHttpGetHeaderByIndex( int     headerIndex,
                                        char *  headerName,
                                        int *   headerNameLth,
                                        char *  headerValue,
                                        int *   headerValueLth
                                       );
```

Diese Funktion hat folgende Funktionsparameter:

>>	`headerIndex`	Index des Header-Feldes, das gelesen werden soll.
<<	`headerName`	Adresse des Puffers, in dem der Name des HTTP-Header-Feldes zurückgegeben wird. Der Puffer muss mindestens `headerNameLth` Bytes lang sein.
<>	`headerNameLth`	Adresse einer Variablen, in der beim Aufruf die Länge des Puffers für den Namen des HTTP-Header-Feldes übergeben und die tatsächliche Länge des Namens zurückgegeben wird.
<<	`headerValue`	Adresse des Puffers, in dem der Wert des HTTP-Header-Feldes zurückgegeben wird. Der Puffer muss mindestens `headerValueLth` Bytes lang sein.
<>	`headerValueLth`	Adresse einer Variablen, in der beim Aufruf die Länge des Puffers für den Wert des HTTP-Header-Feldes übergeben und die tatsächliche Länge des Werts zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_INVALID_INDEX

Der angegebene Index `headerIndex` ist ungültig (kleiner oder gleich 0 oder größer als die Anzahl der im HTTP-Request enthaltenen HTTP-Header-Felder).

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für den Header-Namen `headerName` oder die Adresse der Länge des Header-Namens `headerNameLth` oder die Adresse des Puffers für den Header-Wert `headerValue` oder die Adresse der Länge des Header-Werts `headerValueLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge des Header-Namens `headerNameLth` oder der Wert der Länge des Header-Werts `headerValueLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für den Header-Namen und/oder Header-Wert ist kleiner als die tatsächliche Länge des Namens und/oder Werts. Die Zeichenfolgen werden verkürzt und die tatsächlichen Längen werden zurückgegeben.

8.1.2 Funktion kcHttpGetHeaderByName

Die Funktion `kcHttpGetHeaderByName` liefert den Wert des über den Namen spezifizierten HTTP-Header-Feldes zurück. Das Header-Feld `Authorization` kann nicht gelesen werden.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetHeaderByName

```
kc_http_retcode kcHttpGetHeaderByName( char *  headerName,
                                       char *  headerValue,
                                       int *   headerValueLth
                                     );
```

Diese Funktion hat folgende Funktionsparameter:

>>	`headerName`	Name des HTTP-Header-Feldes, dessen Wert gelesen werden soll.
<<	`headerValue`	Adresse des Puffers, in dem der Wert des HTTP-Header-Feldes zurückgegeben wird. Der Puffer muss mindestens `headerValueLth` Bytes lang sein.
<>	`headerValueLth`	Adresse einer Variablen, in der beim Aufruf die Länge des Puffers für den Wert des HTTP-Header-Feldes übergeben und die tatsächliche Länge des Werts zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_HEADER_NAME_NULL_OR_EMPTY

Die Adresse des Header-Namens `headerName` ist NULL oder der Name ist leer.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für den Header-Wert `headerValue` oder die Adresse der Länge des Header-Werts `headerValueLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge des Header-Werts `headerValueLth` ist kleiner oder gleich 0.

KC_HTTP_HEADER_NOT_FOUND

Es existiert kein HTTP Header mit dem in `headerName` angegebenen Namen.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für den Header-Wert ist kleiner als die tatsächliche Länge des Werts. Der Header-Wert wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.3 Funktion kcHttpGetHeaderCount

Die Funktion `kcHttpGetHeaderCount` liefert die Anzahl der im HTTP-Request enthaltenen Header-Felder zurück. Ein evtl. im HTTP-Request enthaltenes Header-Feld `Authorization` wird dabei nicht mitgezählt, da dieses Header-Feld nicht gelesen werden kann.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetHeaderCount

```
int kcHttpGetHeaderCount( void);
```

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

8.1.4 Funktion kcHttpGetMethod

Die Funktion `kcHttpGetMethod` liefert die HTTP-Methode des HTTP-Requests zurück.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetMethod

```
kc_http_retcode kcHttpGetMethod( char *  methodName,
                                 int *   methodNameLth
                               );
```

Diese Funktion hat folgende Funktionsparameter:

<<	`methodName`	Adresse des Puffers, in dem die Methode des HTTP-Requests zurückgegeben wird. Der Puffer muss mindestens `methodNameLth` Bytes lang sein.
<>	`methodNameLth`	Adresse einer Variablen, in der beim Aufruf die Länge des Puffers für die Methode des HTTP-Requests übergeben und die tatsächliche Länge der Methode zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für die Methode des HTTP-Requests `methodName` oder die Adresse der Länge der Methode `methodNameLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge der Methode `methodNameLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für die Methode des HTTP-Requests ist kleiner als die tatsächliche Länge der Methode. Die Methode wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.5 Funktion kcHttpGetMputMsg

Die Funktion `kcHttpGetMputMsg` gibt die Adresse und die Länge der vom Teilprogramm mit MPUT bereitgestellten Nachricht zurück.

Die Funktion darf nur im HTTP-Exit-Programm beim Bearbeiten der Ausgabenachricht aufgerufen werden.

Funktionsdeklaration kcHttpGetMputMsg

```
kc_http_retcode kcHttpGetMputMsg( void **  mputMessage,
                                  int *    mputMessageLth
                                );
```

Diese Funktion hat folgende Parameter:

<<	`mputMessage`	Adresse einer Variablen, in der die Adresse der MPUT-Nachricht zurückgegeben wird.
<<	`mputMessageLth`	Adresse einer Variablen, in der die Länge der MPUT-Nachricht zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde nicht von einem HTTP-Exit-Programm beim Bearbeiten der Ausgabenachricht aufgerufen.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse der Variablen `mputMessage` oder die Adresse der Variablen `mputMessageLth` ist NULL.

8.1.6 Funktion kcHttpGetPath

Die Funktion `kcHttpGetPath` liefert den normierten Path des HTTP-Requests zurück.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetPath

```
kc_http_retcode kcHttpGetPath( char *  httpPath,
                               int *   httpPathLth
                             );
```

Diese Funktion hat folgende Funktionsparameter:

<<	`httpPath`	Adresse des Puffers, in dem der Path des HTTP-Requests zurückgegeben wird. Der Puffer muss mindestens `httpPathLth` Bytes lang sein.
<>	`httpPathLth`	Adresse einer Variablen, in der die Länge des Puffers für den Path des HTTP-Requests übergeben und die tatsächliche Länge des Path zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für den Path des HTTP-Requests `httpPath` oder die Adresse der Länge des Path `httpPathLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge des Path `httpPathLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für den Path des HTTP-Requests ist kleiner als die tatsächliche Länge des Path. Der Path wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.7 Funktion kcHttpGetQuery

Die Funktion `kcHttpGetQuery` liefert die normierte Query des HTTP-Requests zurück.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetQuery

```
kc_http_retcode kcHttpGetQuery( char *  httpQuery,
                                int *   httpQueryLth
                              );
```

Diese Funktion hat folgende Funktionsparameter:

<<	`httpQuery`	Adresse des Puffers, in dem die Query des HTTP-Requests zurückgegeben wird. Der Puffer muss mindestens `httpQueryLth` Bytes lang sein.
<>	`httpQueryLth`	Adresse einer Variablen, in der die Länge des Puffers für die Query des HTTP-Requests übergeben und die tatsächliche Länge der Query zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für die Query des HTTP-Requests `httpQuery` oder die Adresse der Länge der Query `httpQueryLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge der Query `httpQueryLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für die Query des HTTP-Requests ist kleiner als die tatsächliche Länge der Query. Die Query wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.8 Funktion kcHttpGetRc2String

Die Hilfsfunktion `kcHttpGetRc2String` liefert als Funktionsergebnis den Namen des übergebenen Werts gefolgt von diesem Wert in Klammern als abdruckbaren null-terminierten String zurück.

Funktionsdeklaration kcHttpGetRc2String

```
char * kcHttpGetRc2String( kc_http_retcode httpRc);
```

Diese Funktion hat folgende Funktionsparameter:

>> `httpRc` Wert aus dem `enum kc_http_retcode`.

Beispiel

```
printf("Returncode ist %s.", kcHttpGetRc2String( KC_HTTP_HEADER_NAME_TOO_LONG));
```

Ausgabe

```
Returncode ist KC_HTTP_HEADER_NAME_TOO_LONG (-7).
```

8.1.9 Funktion kcHttpGetReqMsgBody

Die Funktion `kcHttpGetReqMsgBody` gibt die Adresse und die Länge des Message Body des HTTP Requests zurück.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden und im HTTP-Exit-Programm nur beim Bearbeiten der Eingabenachricht aufgerufen werden.

Funktionsdeklaration kcHttpGetReqMsgBody

```
kc_http_retcode kcHttpGetReqMsgBody( void **  reqMessageBody,
                                     int *    reqMessageBodyLth
                                   );
```

Diese Funktion hat folgende Parameter:

<<	`reqMessageBody`	Adresse einer Variablen, in der die Adresse des Message Body des HTTP Requests zurückgegeben wird.
<<	`reqMessageBodyLth`	Adresse einer Variablen, in der die Länge des Message Body des HTTP Requests zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder von einem HTTP-Exit-Programm beim Bearbeiten der Ausgabenachricht aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse der Variablen `reqMessageBody` oder die Adresse der Variablen `reqMessageBodyLth` ist NULL.

8.1.10 Funktion kcHttpGetScheme

Die Funktion `kcHttpGetScheme` liefert das Scheme des HTTP-Requests zurück. Das Scheme wird in der Form zurückgegeben, in der es in der URL enthalten ist, z.B. 'https'.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetScheme

```
kc_http_retcode kcHttpGetScheme( char *  scheme,
                                 int *   schemeLth
                               );
```

Diese Funktion hat folgende Funktionsparameter:

<<	`scheme`	Adresse des Puffers, in dem das Scheme des HTTP-Requests zurückgegeben wird. Der Puffer muss mindestens `schemeLth` Bytes lang sein.
<>	`schemeLth`	Adresse einer Variablen, in der die Länge des Puffers für das Scheme des HTTP-Requests übergeben und die tatsächliche Länge des Scheme zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für das Scheme des HTTP-Requests `scheme` oder die Adresse der Länge des Scheme `schemeLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge des Scheme `schemeLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für das Scheme des HTTP-Requests ist kleiner als die tatsächliche Länge des Scheme. Das Scheme wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.11 Funktion kcHttpGetVersion

Die Funktion `kcHttpGetVersion` liefert die Version des HTTP-Requests zurück. Die Version wird in der Form zurückgegeben, in der sie im HTTP-Protokoll enthalten ist, z.B. 'HTTP/1.1'.

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden.
Im Teilprogramm darf die Funktion nur im ersten Teilprogrammlauf eines Vorgangs aufgerufen werden.

Funktionsdeklaration kcHttpGetVersion

```
kc_http_retcode kcHttpGetVersion( char *  httpVersion,
                                  int *   httpVersionLth
                                );
```

Diese Funktion hat folgende Funktionsparameter:

<<	`httpVersion`	Adresse des Puffers, in dem die Version des HTTP-Requests zurückgegeben wird. Der Puffer muss mindestens `httpVersionLth` Bytes lang sein.
<>	`httpVersionLth`	Adresse einer Variablen, in der die Länge des Puffers für die Version des HTTP-Requests übergeben und die tatsächliche Länge der Version zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT oder nicht im ersten Teilprogrammlauf eines Vorgangs oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers für die Version des HTTP-Requests `httpVersion` oder die Adresse der Länge der Version `httpVersionLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert der Länge der Version `httpVersionLth` ist kleiner oder gleich 0.

KC_HTTP_RESULT_TRUNCATED

Die Länge des Puffers für die Version des HTTP-Requests ist kleiner als die tatsächliche Länge der Version. Die Version wird verkürzt und die tatsächliche Länge wird zurückgegeben.

8.1.12 Funktion kcHttpPercentDecode

Die Funktion `kcHttpPercentDecode` wandelt Zeichen in Prozent-Darstellung, die in der übergebenen Zeichenfolgen enthalten sind, in ihre normale Ein-Zeichen-Darstellung um.

In der URL eines HTTP-Requests können einzelne Zeichen in ihrer %-Darstellung enthalten sein. Dieser Mechanismus ist in RFC 3986 "Uniform Resource Identifier" näher beschrieben. Dabei werden bestimmte US-ASCII Zeichen in ihrer Sedezimal-Darstellung wiedergegeben. Eine solche Prozent-Darstellung wird durch ein vorangestelltes Prozentzeichen ("%") gekennzeichnet; z.B. kann ein Leerzeichen als %20 dargestellt werden oder das Zeichen Tilde ("~") durch die Zeichenfolge %7E.
Die Funktion `kcHttpPercentDecode` kann insbesondere zur Normalisierung der Query eines HTTP-Requests sinnvoll eingesetzt werden.

Funktionsdeklaration kcHttpPercentDecode

```
typedef enum
   {  KC_HTTP_NORM_MINIMUM                                          =  1
    , KC_HTTP_NORM_SPACE_ONLY                                       =  2
    , KC_HTTP_NORM_UNRESERVED                                       =  4
    , KC_HTTP_NORM_RESERVED                                         =  8
    , KC_HTTP_NORM_ALL_BUT_PERCENT                                  = 16
    , KC_HTTP_NORM_ALL                                              = 32
   } kc_http_norm;

kc_http_retcode kcHttpPercentDecode( kc_http_norm  normFlags,
                                     char *        bufferUrl,
                                     int           inputLth,
                                     int *         outputLth
                                   );
```

Diese Funktion hat folgende Funktionsparameter:

>>	`normFlags`	Wert aus `enum kc_http_norm`, mit dem der Umfang der gewünschten Umsetzung festgelegt wird.
<>	`bufferUrl`	Adresse des Puffers, in dem die umzusetzende Zeichenfolge übergeben und die umgesetzte Zeichenfolge zurückgegeben wird. Der Puffer muss mindestens `inputLth` Bytes lang sein.
>>	`inputLth`	Länge der umzusetzenden Zeichenfolge.
<<	`outputLth`	Adresse einer Variablen, in der die Länge der umgesetzten Zeichenfolge zurückgegeben wird.

Mit dem Parameter `normFlags` wird der Umfang der gewünschten Umsetzungen festgelegt.

`normFlags` kann folgende Werte annehmen:

KC_HTTP_NORM_MINIMUM

Bei diesem Wert wird nur eine "case normalization" durchgeführt, d.h. die beiden Buchstaben, die einem %-Zeichen folgen, werden in Großbuchstaben umgesetzt.

KC_HTTP_NORM_SPACE_ONLY

Bei diesem Wert werden zusätzlich zu KC_HTTP_NORM_MINIMUM alle "%20" Codierungen durch Leerzeichen ersetzt.

KC_HTTP_NORM_UNRESERVED

Bei diesem Wert werden zusätzlich zu KC_HTTP_NORM_SPACE_ONLY alle Prozent-Darstellungen von Unreserved Characters durch ihre "normale" Ein-Zeichen Darstellung ersetzt. Zur Definition von Unreserved Characters siehe RFC 3986 "Uniform Resource Identifier".

KC_HTTP_NORM_RESERVED

Bei diesem Wert werden zusätzlich zu KC_HTTP_NORM_UNRESERVED alle Prozent-Darstellungen von Reserved Characters durch ihre "normale" Ein-Zeichen Darstellung ersetzt. Bzgl. der Definition von Reserved Characters siehe RFC 3986 "Uniform Resource Identifier".

KC_HTTP_NORM_ALL_BUT_PERCENT

Bei diesem Wert werden zusätzlich zu KC_HTTP_NORM_RESERVED alle Prozent-Darstellungen außer %25 ("%") durch ihre "normale" Ein-Zeichen Darstellung ersetzt.

KC_HTTP_NORM_ALL

Bei diesem Wert werden alle Prozent-Darstellungen durch ihre "normale" Ein-Zeichen Darstellung ersetzt.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse des Puffers `bufferUrl` oder die Adresse für die Länge der umgesetzten Zeichenfolge `outputLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Die Länge der umzusetzenden Zeichenfolge `inputLth` ist negativ oder die umzusetzende Zeichenfolge enthält ein Nullzeichen (\0).

KC_HTTP_NORMALIZATION_ERROR

Im Parameter `normFlags` wurde ein ungültiger Wert übergeben oder bei der Normalisierung trat ein Fehler auf.

KC_HTTP_PARAM_INVALID_URLSTRING

Die umzusetzende Zeichenfolge enthält ein Prozentzeichen ("%"), aber eines der nächsten beiden Zeichen ist keine abdruckbare Hexadezimal-Zahl.

8.1.13 Funktion kcHttpPutHeader

Mit der Funktion `kcHttpPutHeader` kann der Name und Wert eines HTTP-Header-Feldes für die HTTP Response übergeben werden.
Die Länge des Header-Namens und die Länge des Header-Werts sind auf jeweils 256 Bytes beschränkt. Die Anzahl der Header-Felder für eine HTTP-Response ist auf 100 beschränkt. Die Gesamtlänge der Header einschließlich der Steuerzeichen (CLRF) ist auf 7000 Bytes beschränkt.
Der Header-Name kann in beliebiger Groß-/Kleinschreibung übergeben werden und wird so gesendet, wie er übergeben wurde. Es wird nicht erkannt, wenn der gleiche Header mehrfach übergeben wird. Die Werte werden daher nicht zusammengefasst, sondern alle Header werden einzeln gesendet.

> Einige HTTP-Header-Felder werden immer von openUTM gesetzt und dürfen nicht vom Anwendungsprogramm übergeben werden. Andere HTTP-Header-Felder werden von openUTM nur dann gesetzt, wenn sie nicht vom Anwendungsprogramm übergeben werden.
> Details finden Sie im Kapitel „Versorgung von HTTP-Headern".

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden. Im HTTP-Exit-Programm darf die Funktion nur beim Bearbeiten der Ausgabenachricht aufgerufen werden.

Wenn die Funktion im Teilprogrammlauf aufgerufen wird, so muss im selben Teilprogrammlauf mit MPUT eine Ausgabenachricht an den HTTP-Client bereitgestellt und der Vorgang beendet werden, ansonsten geht die übergebene Header-Information verloren.

Funktionsdeklaration kcHttpPutHeader

```
kc_http_retcode kcHttpPutHeader( char *  headerName,
                                 char *  headerValue
                               );
```

Diese Funktion hat folgende Funktionsparameter:

>> `headerName` Adresse des Namens des HTTP-Header-Feldes für die HTTP Response.

>> `headerValue` Adresse des Werts des HTTP-Header-Feldes für die HTTP Response.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT des Teilprogramms oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client aufgerufen.

KC_HTTP_HEADER_NAME_NULL_OR_EMPTY

Die Adresse des Header-Namens `headerName` ist NULL oder der Name ist leer.

KC_HTTP_HEADER_VALUE_NULL_OR_EMPTY

Die Adresse des Header-Werts `headerValue` ist NULL oder der Wert ist leer.

KC_HTTP_HEADER_NAME_TOO_LONG

Der Name des angegebenen Header-Feldes ist zu lang.

KC_HTTP_HEADER_VALUE_TOO_LONG

Der Wert des angegebenen Header-Feldes ist zu lang.

KC_HTTP_HEADER_NOT_ALLOWED

Der angegebene Header-Name darf nicht gesetzt werden.

KC_HTTP_MAX_HEADER_COUNT

Die maximale Anzahl der Header wurde überschritten.

KC_HTTP_MAX_HEADER_TOTAL_LENGTH

Die maximale Gesamtlänge der Header einschließlich Steuerzeichen wurde überschritten.

KC_HTTP_HEADER_NAME_NOT_PRINTABLE

Der Name des angegebenen Header-Feldes enthält ein nicht abdruckbares Zeichen.

KC_HTTP_HEADER_VALUE_NOT_PRINTABLE

Der Wert des angegebenen Header-Feldes enthält ein nicht abdruckbares Zeichen.

8.1.14 Funktion kcHttpPutMgetMsg

Mit der Funktion `kcHttpPutMgetMsg` kann eine Nachricht für das Teilprogramm von einem HTTP-Exit-Programm erzeugt werden, die mit MGET gelesen werden kann.
Die Nachricht kann sukzessive mit mehreren Aufrufen der Funktion aufgebaut werden, ohne dass sich der Aufrufer selbst um die Anforderung und Verwaltung des benötigten Speicherbereichs kümmern muss. Wie Sie diese Nachricht in Nachrichtenteile strukturieren, die im Teilprogramm mit MGET NT gelesen werden können, ist in Kapitel „Event-Exit HTTP" beschrieben.

Diese Funktion darf nur von einem HTTP-Exit-Programm beim Bearbeiten der Eingabenachricht aufgerufen werden.

Funktionsdeklaration kcHttpPutMgetMsg

```
kc_http_retcode kcHttpPutMgetMsg( void *   mgetMessage,
                                  int      mgetMessageLth,
                                  void **  totalMgetMessage,
                                  int *    totalMgetMessageLth
                                );
```

Diese Funktion hat folgende Parameter:

>>	`mgetMessage`	Adresse der übergebenen Nachricht bzw. des übergebenen Nachrichtenteils.
>>	`mgetMessageLth`	Länge der übergebenen Nachricht bzw. des übergebenen Nachrichtenteils.
<<	`totalMgetMessage`	Adresse einer Variablen, in der die Adresse der bisher aufgebauten Gesamtnachricht zurückgegeben wird.
<<	`totalMgetMessageLth`	Adresse einer Variablen, in der die Länge der bisher aufgebauten Gesamtnachricht zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde nicht von einem HTTP-Exit Programm beim Bearbeiten der Eingabenachricht aufgerufen.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse der Nachricht `mgetMessage`, die Adresse der Variablen `totalMgetMessage` oder die Adresse der Variablen `totalMgetMessageLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert des Parameters `mgetMessageLth` ist kleiner oder gleich 0.

KC_HTTP_MEMORY_INSUFFICIENT

Es konnte kein weiterer Speicher für die Gesamtnachricht allokiert werden.

8.1.15 Funktion kcHttpPutRspMsgBody

Mit der Funktion `kcHttpPutRspMsgBody` kann der Message Body für die HTTP-Response bereitgestellt werden. Der Message Body kann sukzessive mit mehreren Aufrufen der Funktion aufgebaut werden, ohne dass sich der Aufrufer selbst um die Anforderung und Verwaltung des benötigten Speicherbereichs kümmern muss.

Diese Funktion darf nur von einem HTTP-Exit-Programm beim Bearbeiten der Ausgabenachricht aufgerufen werden.

Funktionsdeklaration kcHttpPutRspMsgBody

```
kc_http_retcode kcHttpPutRspMsgBody( void *   rspMessageBody,
                                     int      rspMessageBodyLth,
                                     void **  totalRspMessageBody,
                                     int *    totalRspMessageBodyLth
                                   );
```

Diese Funktion hat folgende Parameter:

>>	`rspMessageBody`	Adresse des übergebenen (Teils des) Message Body.
>>	`rspMessageBodyLth`	Länge des übergebenen (Teils des) Message Body.
<<	`totalRspMessageBody`	Adresse einer Variablen, in der die Adresse des bisher aufgebauten gesamten Message Body zurückgegeben wird.
<<	`totalRspMessageBodyLth`	Adresse einer Variablen, in der die Länge des bisher aufgebauten gesamten Message Body zurückgegeben wird.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde nicht von einem HTTP-Exit Programm beim Bearbeiten der Ausgabenachricht aufgerufen.

KC_HTTP_PARAM_VALUE_NULL

Die Adresse der Nachricht `rspMessageBody`, die Adresse der Variablen `totalRspMessageBody` oder die Adresse der Variablen `totalRspMessageBodyLth` ist NULL.

KC_HTTP_INVALID_LENGTH

Der Wert des Parameters `rspMessageBodyLth` ist kleiner oder gleich 0.

KC_HTTP_MEMORY_INSUFFICIENT

Es konnte kein weiterer Speicher für den Message Body der HTTP Response allokiert werden.

8.1.16 Funktion kcHttpPutStatus

Mit der Funktion `kcHttpPutStatus` kann ein Status-Code für die HTTP Response übergeben werden.
Zusätzlich kann eine Reason-Phrase angegeben werden.
Die Länge der Reason-Phrase ist auf 256 Bytes beschränkt. Wird als zweiter Parameter NULL übergeben, so setzt openUTM die Standard-Reason-Phrase für den gesetzten Status-Code ein.
Folgende Status-Codes dürfen nicht gesetzt werden:

- 100 Continue
- 101 Switching Protocols
- 203 Non-Authoritative Information
- 301 Moved Permanently
- 302 Found
- 307 Temporary Redirect
- 401 Unauthorized
- 407 Proxy Authentication Required
- 408 Request Timeout
- 411 Length Required
- 413 Request Entity Too Large
- 417 Expectation Failed
- 502 Bad Gateway
- 504 Gateway Timeout
- 505 HTTP Version Not Supported

Die Funktion kann im Teilprogramm und im HTTP-Exit-Programm aufgerufen werden. Im HTTP-Exit-Programm darf die Funktion nur beim Bearbeiten der Ausgabenachricht aufgerufen werden. Wenn die Funktion im Teilprogrammlauf aufgerufen wird, so muss im selben Teilprogrammlauf mit MPUT eine Ausgabenachricht an den HTTP-Client bereitgestellt und der Vorgang beendet werden, ansonsten geht die übergebene Header-Information verloren.

Funktionsdeklaration kcHttpPutStatus

```
kc_http_retcode kcHttpPutStatus( int     statusCode,
                                 char *  reasonPhrase
                               );
```

Diese Funktion hat folgende Funktionsparameter:

>> `statusCode` Status-Code für die HTTP Response.

>> `reasonPhrase` Adresse des Puffers mit der Reason-Phrase oder NULL.

Der zurückgelieferte Funktionswert hat folgende Bedeutung:

KC_HTTP_OK

Die Funktion wurde erfolgreich ausgeführt.

KC_HTTP_FUNCTION_CALL_NOT_ALLOWED

Die Funktion wurde vor dem KDCS-Aufruf INIT des Teilprogramms oder außerhalb eines HTTP-Exit-Programms aufgerufen.

KC_HTTP_NO_HTTP_CLIENT

Der Vorgang wurde nicht von einem HTTP-Client gestartet.

KC_HTTP_INVALID_STATUS_CODE

Der angegebene Status-Code ist ungültig.

KC_HTTP_STATUS_CODE_NOT_ALLOWED

Der angegebene Status-Code darf nicht übergeben werden.

KC_HTTP_REASON_PHRASE_TOO_LONG

Die angegebene Reason-Phrase ist zu lang.

KC_HTTP_REASON_PHRASE_NOT_PRINTABLE

Die Reason-Phrase enthält ein nicht abdruckbares Zeichen.

8.2 Adressierung TAC

Ein HTTP-Client kann einen Service einer UTM-Anwendung auf verschiedene Weise adressieren. Zum einen kann der TAC des gerufenen Service direkt im Path der URL angegeben werden. Alternativ kann über die UTM-Generierung eine Abbildung von Path-Angaben in einer URL auf einen TAC festgelegt werden. Die Zuordnung eines Path zu einem TAC der UTM-Anwendung wird mit Hilfe der KDCDEF Anweisung HTTP-DESCRIPTOR gesteuert.

Beim Empfang eines HTTP-Requests in einer UTM-Anwendung erfolgt die Auswahl des TACs in folgender Reihenfolge:
openUTM sucht den Path eines HTTP-Requests zunächst in den HTTP-Deskriptoren, die dem BCAMAPPL zugeordnet sind, über den der Request empfangen wurde. Wird der Path dort nicht gefunden, dann durchsucht openUTM als nächstes die HTTP-Deskriptoren, die mit BCAMAPPL=*ALL generiert sind. Falls der Path auch dort nicht gefunden wird, dann prüft openUTM, ob der Path einem generierten TAC entspricht. Trifft auch dies nicht zu, prüft openUTM, ob ein Path mit "/*" generiert ist, zunächst für den BCAMAPPL, über den der Request empfangen wurde, dann für BCAMAPPL=*ALL.

8.3 Bewertung der HTTP-Header eines HTTP-Requests

openUTM bewertet in einem HTTP-Request die im Folgenden aufgeführten Header-Felder. Sind im HTTP-Request weitere HTTP-Header enthalten, so werden diese von openUTM nicht bewertet. Ein Anwendungsprogramm kann diese Header aber mit den Funktionen `kcHttpGetHeaderByIndex` bzw. `kcHttpGetHeaderByName` lesen und auswerten.

In der folgenden Liste der von openUTM bewerteten Header-Felder sind die Header-Felder, die in einem HTTP-Request enthalten sein können, als "Optional", und die Header-Felder, die in einem HTTP-Request enthalten sein müssen, als "Erforderlich" gekennzeichnet:

- **Accept**
  Optional
  Mit diesem Header drückt ein Client seinen Wunsch bzgl. der Darstellung der Antwortnachricht aus.
  openUTM dient dieser Header zur Aufbereitung der Ausgabenachricht, falls kein HTTP-Deskriptor bzw. ein HTTP-Deskriptor mit Angabe von HTTP-EXIT=*SYSTEM gefunden wurde. Der Header-Wert wird in folgender Reihenfolge nach diesen Mustern durchsucht:
  1. *text/html*
  2. *text/plain*
  3. *text/** (Wirkung wie *text/html*)
  4. *application/octet-stream*

  Wird keiner dieser Werte gefunden, dann wird die Ausgabenachricht wie bei *text/html* behandelt.
- **Accept-Charset**
  Optional
  Mit diesem Header drückt ein Client seinen Wunsch bzgl. der Codierung der Antwortnachricht aus.
  BS2000-Systeme:
  Der Header-Wert wird nach einem generierten CHAR-SET Namen durchsucht. Der erste gefundene CHAR-SET Name bestimmt die Code-Konvertierung der Ausgabenachricht. Wird kein CHAR-SET Name gefunden, dann wird die Standard-Konvertierung von openUTM angewendet.
  Unix-, Linux- und Windows-Systeme:
  Keine Bewertung durch openUTM. Auf Unix-, Linux- und Windows-Systemen erfolgt grundsätzlich keine Code-Konvertierung der Ausgabenachricht.
- **Authorization**
  Optional
  In diesem Header kann ein Client Authentisierungsdaten an openUTM übegeben.
  Ist dieser Header angegeben, dann muss der Wert mit “basic “ beginnen. <userid>:<password> muss base64 codiert sein. Andernfalls wird der Request mit Status-Code "400 invalid http header format“ zurückgewiesen.
- **Connection**
  Optional
  Mit diesem Header kann ein Client mitteilen, ob der Server die Verbindung nach Senden der Antwortnachricht abbauen soll oder nicht.
  Es wird nur der Header-Wert „keep-alive“ bewertet. Falls dieser Wert angegeben ist, baut openUTM nach dem Senden der Ausgabenachricht die Verbindung zum Client nicht ab. Andernfalls wird die Verbindung nach dem Senden der HTTP-Response von openUTM abgebaut.

- **Content-Length**
  Erforderlich, wenn der Request einen Message Body enthält.
  Dieser Header muss vom HTTP-Client mit der korrekten Länge des Message Body versorgt werden.
  Wird ein HTTP-Request mit Message Body und ohne Angabe eines Content-Length Headers empfangen, dann wird der Message Body ignoriert.
- **Content-Type**
  Optional
  Dieses Header-Feld beschreibt den MIME-Type der im Message Body enthaltenen Nachricht.
  BS2000-Systeme:
  Falls zum Path ein HTTP-Deskriptor gefunden wird, der mit CONVERT-TEXT=*YES generiert ist, so bestimmt ein nach „charset=“ angegebener generierter CHAR-SET Name die Code-Konvertierung der Eingabenachricht, sofern nicht APPLICATION/OCTET-STREAM gefunden wird. Wird kein CHAR-SET Name gefunden, so erfolgt die Standard-Konvertierung der Eingabenachricht.
  Unix-, Linux- und Windows-Systeme:
  Keine Bewertung durch openUTM. Auf Unix-, Linux- und Windows-Systemen erfolgt grundsätzlich keine Code-Konvertierung der Eingabenachricht.
- **Expect**
  Optional
  Mit diesem Header kann ein Client den Server auffordern die Korrektheit des gesendeten HTTP-Request zu prüfen, bevor der Client den Message Body sendet.
  Der Header-Wert muss “100-continue” sein. In diesem Fall darf der HTTP-Client vor dem Senden des Message Body auf die Antwort von openUTM mit Status Code 100 warten. Ist der Header-Value nicht „100-continue“, so wird der HTTP-Request mit Status-Code "417 expectation failed“ zurückgewiesen.
- **Host**
  Erforderlich
  Dieser Header muss vom HTTP-Client mit dem Namen des HTTP-Servers versorgt werden.
- **Transfer-Encoding**
  Optional
  Dieser Header beschreibt die Transformationen, die angewendet wurden, um den Inhalt sicher zum Server zu transportieren.
  Ist dieser Header angegeben, dann muss der Wert “identity” sein. Andernfalls wird der Request mit Status-Code "501 transfer encoding rejected“ zurückgewiesen.

8.4 Versorgung der HTTP-Header einer HTTP-Response

openUTM versorgt in einer HTTP-Response eine Reihe von Header-Feldern. Dabei wird unterschieden zwischen Header-Felder, die vom Anwender nicht versorgt werden dürfen und Header-Felder, die openUTM nur dann versorgt werden, wenn sie vom Anwender nicht versorgt wurden.

Liste der von openUTM versorgten Header-Felder in einer HTTP-Response, die vom Anwender nicht versorgt werden dürfen:

- **Connection**
  Angabe, ob die Verbindung vom HTTP-Server nach dem Senden der HTTP-Response abgebaut wird. openUTM versorgt dieses Header-Feld immer mit dem Wert „close“, sofern der Client nicht im HTTP-Request Header Connection „keep-alive“ angegeben hat.
- **Content-Length**
  Länge des Message Bodys.
- **Date**
  Datum und Zeit zum Sendezeitpunkt.
  Beispiel: Mon, 28 Oct 2019 14:38:45 GMT
- **Server**
  Angaben zum UTM-HTTP-Server in der Form: “UTM HTTP-SERVER“, openUTM Version, Anwendungsname, Plattform.
  Beispiel: UTM HTTP-SERVER V07.0A00 SAMPLE Linux Intel

Liste der Header-Felder in einer HTTP-Response, die von openUTM nur dann versorgt werden, wenn sie nicht vom Anwender versorgt wurden:

- **Cache-Control**
  Angabe, ob und wie lange die Antwort vom Client oder von Gateway-Rechnern gespeichert und als Antwort für nachfolgende identische Requests wiederverwendet werden darf.
  Von openUTM wird dieses Header-Feld standardmäßig mit dem Wert „no-cache, no-store, must-revalidate“ versorgt.

- **Content-Type**
  Angabe des MIME Typs und ggf. Zeichensatzes des Message Bodys.
  openUTM versorgt dieses Header-Feld folgendermaßen:
  - *text/html;charset=*
    - wenn die Auswahl des durch den HTTP Request gestarteten TAC ohne HTTP-Deskriptor oder über einen HTTP-Deskriptor mit HTTP-EXIT=*SYSTEM erfolgte
      und
    - der Request das HTTP-Header-Feld Accept nicht enthielt oder der Wert des Headers *text/html* oder *text/*,* aber nicht *text/plain* oder weder *text/...* noch *application/octet-stream* enthielt.
  - *text/plain;charset=*
    wenn die Auswahl des durch den HTTP Request gestarteten TAC
    - über einen HTTP-Deskriptor mit HTTP-EXIT ungleich *SYSTEM und CONVERT-TEXT = *YES erfolgte oder
    - über keinen HTTP-Deskriptor oder einen HTTP-Deskriptor mit HTTP-EXIT=*SYSTEM erfolgte und der Request das HTTP-Header-Feld Accept mit dem Wert *text/plain* aber nicht *text/html* enthielt.
  - *application/octet-stream*
    wenn die Auswahl des durch den HTTP Request gestarteten TAC
    - über einen HTTP-Deskriptor mit HTTP-EXIT ungleich *SYSTEM und CONVERT-TEXT = *NO erfolgte oder
    - über keinen HTTP-Deskriptor oder einen HTTP-Deskriptor mit HTTP-EXIT=*SYSTEM erfolgte und der Request das HTTP-Header-Feld Accept mit dem Wert *application/octet-stream* aber nicht *text/...* enthielt.
  - *charset=*
    Hat das HTTP-Header-Feld Content-type den Wert *text/html* oder *text/plain,* so wird der Parameter *charset* wie folgt versorgt:

    Unix-, Linux- und Windows-Systeme:
    charset=ISO-8859-1

    BS2000-Systeme:
    charset=<charset-name>
    - wenn die Auswahl des durch den HTTP Request gestarteten TAC ohne HTTP-Deskriptor oder über einen HTTP-Deskriptor mit CONVERT-TEXT=*YES erfolgte und der Request das HTTP-Header-Feld Accept-Charset und der Wert des Headers einen mit CHAR-SET generierten Namen <*charset-name*> enthielt.
    - wenn die Auswahl des durch den HTTP Request gestarteten TAC über einen HTTP-Deskriptor mit CONVERT-TEXT=*YES erfolgte und der Request den Header Content-Type und der Wert des Header einen mit CHAR-SET generierten Namen <*charset-name*> enthielt und der Request das HTTP-Header-Feld *Accept* nicht oder der Wert des Headers keinen mit CHAR-SET generierten Namen enthielt.

    charset=ISO-8859-1

    sonst
- **X-Content-Type-Options**
  Verbietet MIME-Sniffing.
  openUTM versorgt dieses Header-Feld mit dem Wert „nosniff“
- **X-Frame-Options**
  Schutz vor Clickjacking.
  openUTM versorgt dieses Header-Feld mit dem Wert „deny“

- **X-Xss-Protection**
  Filter für Cross-Site-Scripting.
  openUTM versorgt dieses Header-Feld mit dem Wert „1“.

8.5 Besonderheiten bei der Kommunikation mit HTTP-Clients

Nachfolgend aufgelistete Besonderheiten sind bei der Kommunikation mit HTTP-Clients zu beachten:

1. HTTP-Clients dürfen nur Einschritt-Dialog-Vorgänge aufrufen.
2. Es gelten folgende Restriktionen für HTTP-Nachrichten:
   a. Die Request Line sowie die Status Line dürfen jeweils maximal 2KB groß sein.
   b. Die Länge aller Header darf nicht größer sein als 8KB.
   c. Eine HTTP-Nachricht darf maximal 100 Header enthalten.
   d. Die Länge einer Nachricht ist auf maximal 32.700 Bytes beschränkt.
   e. Die Länge des Message Bodys ist auf 32000 Bytes beschränkt.
3. Für HTTP-Clients darf kein Anmeldevorgang konfiguriert werden.
4. Bei der Behandlung von K-Meldungen wird für HTTP-Clients das Meldungsziel PARTNER nicht bewertet. Stattdessen werden K-Meldungen an HTTP-Clients unabhängig von den konfigurierten Meldungszielen nur genau dann ausgegeben, wenn eine K-Meldung als Dialogantwort an den HTTP-Client benötigt wird.
   Beispiele hierfür sind die Meldungen K017, die bei einem Vorgangsabbruch an einen Client gesendet wird, und K034, die beim Rücksetzen einer verteilten Transaktion gesendet wird.
5. openUTM verwendet keine Cookies.
6. An HTTP-Clients können keine Asynchron-Nachrichten gesendet werden.
7. Für HTTP-Clients ist kein Bildschirmwiederanlauf (KDCDISP, KDCLAST) möglich.

8.6 Verhalten in Fehlersituationen

- Der Empfang eines HTTP-Requests, der andere HTTP-Methoden als POST, GET, PUT und DELETE enthält, wird von openUTM mit HTTP Status-Code "501 Method Not Implemented" abgewiesen.
- Falls einem HTTP-Request kein TAC zugeordnet werden kann, wird er von openUTM mit HTTP Status-Code „404 Not Found“ abgewiesen.
- Grundsätzlich gilt beim Empfang eines ungültigen HTTP-Requests, dass die Verbindung zum HTTP-Client abgebaut wird, nachdem eine negative HTTP-Response gesendet wurde.

Eine Übersicht über die von openUTM in Fehlersituationen verwendeten HTTP Status-Codes finden Sie im Kapitel "HTTP Status-Codes“ im openUTM-Handbuch „Meldungen, Test und Diagnose“.

9 Event-Funktionen

Um auf bestimmte Ereignisse per Programm reagieren zu können, bietet openUTM die Möglichkeit, Event-Funktionen zu verwenden. Im Gegensatz zu "normalen" Teilprogrammen, die durch Angabe eines Transaktionscode aufgerufen werden, startet openUTM diese Teilprogramme bei bestimmten Ereignissen.

Es gibt zwei unterschiedliche Arten von Event-Funktionen:

- Event-Exits, welche **keine KDCS-Aufrufe** enthalten **dürfen** und
- Event-Services, die **KDCS-Aufrufe** enthalten **müssen**.

Hinweis zu BS2000-Systemen

Da auch STXIT-Routinen, wie sie auf BS2000-Systemen verwendet werden, ereignisgesteuert sind, werden sie in diesem Kapitel mit beschrieben, und zwar im Abschnitt „STXIT-Routinen (BS2000-Systeme)“. Auf die Ereignisbehandlung in ILCS-Programmen wird in Abschnitt „Ereignisbehandlung in ILCS-Programmen (BS2000-Systeme)“ eingegangen.

Event-Exits:

INPUT	Dieser Event-Exit wird bei Eingabe von einem Terminal aufgerufen.
HTTP	Dieser Event-Exit wird beim Lesen der Eingabe-Nachricht von einem sowie vor dem Senden der HTTP-Response an einen HTTP-Client aufgerufen.
START	Diese Event-Exits (maximal 8) werden bei jedem Start des Anwendungsprogramms aufgerufen.
SHUT	Diese Event-Exits (maximal 8) werden bei jeder Beendigung des Anwendungsprogramms aufgerufen, auch bei PEND ER.
VORGANG	Dieser Event-Exit wird bei Vorgangsbeginn und Vorgangsende aufgerufen, auch bei fehlerhaftem Ende oder Ende wegen Verbindungsverlust, und bei Vorgangs-Wiederanlauf.
FORMAT	Nur auf BS2000-Sysemen: Dieser Event-Exit wird aufgerufen, wenn eine Nachricht ein- bzw. ausgegeben wird, die Sie mit einer eigenen Formatierungsroutine formatieren möchten (-Format).

Event-Services:

BADTACS	Dieser Dialog-Vorgang wird aufgerufen, wenn von einem Terminal oder einer TS-Anwendung ein ungültiger Transaktionscode angegeben wurde, oder der Datenschutz verletzt wurde.
MSGTAC	Dieser Asynchron-Vorgang wird aufgerufen, wenn openUTM eine Meldung ausgibt, für die als Meldungsziel MSGTAC definiert wurde (siehe openUTM-Handbuch „Meldungen, Test und Diagnose “, Ändern von Meldungen).
SIGNON	Dieser Dialog-Vorgang wird immer dann aufgerufen, wenn sich ein Terminal-Benutzer, eine TS-Anwendung oder ein UPIC-Client an die Anwendung anmeldet. Voraussetzung ist, dass für den Transportsystem-Zugriffspunkt, über den sich der Client anmeldet, ein Anmelde-Vorgang generiert ist; für UPIC-Clients muss zusätzlich in der SIGNON-Anweisung UPIC=YES generiert sein. Für jeden Transportsystem-Zugriffspunkt einer UTM-Anwendung (generiert mit MAX APPLINAME bzw. BCAMAPPL) kann ein eigener Anmelde-Vorgang generiert werden (siehe Parameter SIGNON-TAC der Anweisung BCAMAPPL im openUTM-Handbuch „Anwendungen generieren“).

Datenbankaufrufe sind nur in den Teilprogrammen für VORGANG, BADTACS, MSGTAC und SIGNON erlaubt, bei den Teilprogrammen für INPUT, START und SHUT hingegen nicht.

Nur auf BS2000-Systemen: In den Teilprogrammen für FORMAT sind keine Datenbankaufrufe erlaubt.

9.1 Event-Exits

Event-Exits werden als Unterprogramme ohne KDCS-Aufrufe erstellt. In diesem Abschnitt werden folgende Event-Exits beschrieben:

INPUT
HTTP
START
SHUT
VORGANG
FORMAT (nur auf BS2000-Systemen)

Die Event-Exits INPUT, START, SHUT und FORMAT werden mit einer KDCDEF-Anweisung EXIT definiert. Den Event-Exit VORGANG definieren Sie mit TAC- und PROGRAM-Anweisung.

Bei den Event-Exits sind Datenbank-Aufrufe nur im VORGANG-Exit erlaubt.

Lesen des Anwendungsnamens

Ein Event-Exit kann den Namen der eigenen Anwendung lesen, indem er den Entry KDCAPLI aufruft, siehe folgendes Beispiel:

Assembler-Programm

```
         EXTRN  KDCAPLI
AKDCAPLI DC     A(KDCAPLI)
```

C-Programm

```
extern char KDCAPLI[8];
```

9.1.1 Event-Exit INPUT

Mit diesem Event-Exit kann die Wirkung einer Terminal-Eingabe bestimmt werden.

openUTM ruft den Event-Exit INPUT - mit einigen Ausnahmen - bei jeder Eingabe von einem Terminal auf. Diese Ausnahmen sind:

- Eingaben, wenn noch kein Benutzer angemeldet ist
- Eingaben im Event-Service SIGNON
- Eingaben mittels einer mit SFUNC generierten Funktionstaste
- Eingaben, nachdem ein Programm den Aufruf SIGN OB/OF gegeben hat

openUTM übergibt dem Programm für einen INPUT-Exit die Adresse eines Parameterbereichs, welcher aus einem Eingabebereich und einem Ausgabebereich besteht.

Für die Deklaration des Parameterbereichs stehen Ihnen Sprach-spezifische Datenstrukturen zur Verfügung, für COBOL das COPY-Element KCINPC, für C/C++ die Include-Datei *kcinp.h*. Die einzelnen Felder des Parameterbereichs sowie die Bedeutung der einzelnen Felder sind auf den folgenden Seiten ab Tablelle „Parameterbereich KDCINPC/kdcinp.h" beschrieben.

Auf BS2000-Systemen übergibt openUTM dem Input-Exit zusätzlich einen zweiten Parameter, der weitere Eingabefelder enthält. Auch für die Strukturierung des 2. Parameters steht eine Sprach-spezifische Datenstruktur zur Verfügung, für COBOL das COPY-Element KCCFC und für C/C++ die Include-Datei *kccf.h.* Der 2. Parameterbereich ist auf den Seiten ab Abschnitt „Zweiter Parameterbereich KCCFC/kccf.h (für BS2000-Systeme)" beschrieben.

Das Programm analysiert die in den Eingabefeldern eingetragenen Werte und versorgt daraufhin den Ausgabebereich. Aufgrund der Werte, die im Ausgabebereich eingetragen wurden, entscheidet openUTM, welche der folgende Aktionen ausgeführt werden:

- Fortsetzen eines Vorgangs
- Starten eines neuen Vorgangs
- Kellern eines Vorgangs und Starten eines neuen Vorgangs
- Ausführen eines Benutzer-Kommandos, z.B. KDCOFF
- Senden einer Meldung mit Fehlerinformation an das Terminal

Anwendungsmöglichkeiten

Der Eingabe-Exit bietet für die Terminalschnittstelle zusätzliche Freiheiten bei der Gestaltung der Benutzeroberfläche.

- Position von Transaktionscodes oder KDC-Kommandos in der Nachricht:Transaktionscode oder KDC-Kommando müssen nicht am Anfang der Nachricht stehen.
- Sichtbarkeit der Transaktionscodes am Bildschirm:
  Es ist nicht notwendig, dass ein Transaktionscode auf dem Bildschirm sichtbar wird, wenn man einen Vorgang starten will.
  Beispielsweise kann ein Menü mehrere Tätigkeiten anbieten. Der Terminal-Benutzer, wählt durch Eingabe eines Textes oder einer Nummer eine von ihnen aus. Diese Eingabe muss noch kein Transaktionscode sein, erst der INPUT-Exit setzt sie in einen Transaktionscode (Vorgangs-TAC) um. Damit werden die Transaktionscodes, die ja eine Generierungsinformation sind, von der Dialog-Oberfläche entkoppelt.

- Darstellung von Kommando-Namen:
  Es ist nicht notwendig, die Schreibweise "KDC..." zu verwenden, um die Wirkung eines KDC-Kommandos auszulösen.
  Wenn es zweckmäßig ist, kann man eine andere Darstellung wählen, z.B. "/" an Stelle von "KDC".

Parameterbereich KDCINPC/kdcinp.h	
Eingabebereich (wird von openUTM versorgt)	
Feldname	**Inhalt**
KCIFCH	erste 8 Zeichen der Eingabe
KCIMF/kcifn	Formatkennzeichen
KCICVTAC	Transaktionscode des Vorgangs
KCICVST	Vorgangs-Status
KCIFKEY	Wert der F-Taste: 1,...,20 / binär null
KCIKKEY	binär null / auf BS2000-Systemen: Wert der K-Taste: 1,...,20
KCICFINF: "NO"/"MO"/"ON"/"UN" [1]	Informationen über die Formatierung
KCILTERM	aktueller LTERM-Partner
KCIUSER	aktuelle Benutzerkennung
Ausgabebereich (vom INPUT-Exit zu versorgen)	
Feldname	**Inhalt**
KCINTAC bzw. KCINCMD	nächster Vorgangs-TAC bzw. nächstes Benutzerkommando
KCICCD	Code für die Wirkung der Eingabe: "ER"/"CC"/"SC"/"ST"/"CD"
KCICUT	TAC abschneiden: "Y"/"N"
KCIERRCD	Fehlerinfo für das Terminal (4 Byte)

[1] "MO" / "ON" / "UN" (nur auf BS2000-Systemen)

In den Eingabebereich trägt openUTM folgende Werte ein:

KCIFCH

In das Feld KCIFCH die ersten 8 Zeichen der Eingabe, höchstens jedoch bis zum ersten Leerzeichen.

KCIMF/kcifn

In das Feld KCIMF/kcifn das Formatkennzeichen:

- Leerzeichen im Zeilenmodus oder
- Formatkennzeichen des Formats, das am Bildschirm steht (nur auf BS2000-Systemen).

Nur auf BS2000-Systemen:
Das Feld KCIMF/kcifn kann auch das Formatkennzeichen #!POPUP enthalten. #!POPUP bedeutet, dass eine Box am Bildschirm steht. Über die FHS-Service-Funktion KDCFHS mit dem USER-CODE INFD können Sie die Namen aller Formate erhalten, die am Bildschirm stehen.

KCICVTAC

In das Feld KCICVTAC den Transaktionscode, mit dem der aktuelle Vorgang gestartet wurde (wenn man sich innerhalb eines Vorgangs befindet).

KCICVST

In das Feld KCICVST den Vorgangs-Status:

- ES (**E**nd of dialog **S**tep)
  Ende des Verarbeitungsschritts mit PEND KP oder PGWT KP,
- ET (**E**nd of **T**ransaction)
  Ende der Transaktion mit PEND RE,
- RS (**R**eturn from **S**tack)
  Ende des eingeschobenen Vorgangs, die Eingabe ist für den gekellerten Vorgang bestimmt.
- EC (**E**nd of **C**onversation)
  Der letzte Verarbeitungsschritt wurde mit PEND FI beendet, die Eingabe ist für einen neuen Vorgang bestimmt.

KCIFKEY

den Wert der F-Taste (1 - 20, auf BS2000-Systemen: 1 - 24), falls betätigt, sonst binär null.

KCIKKEY

den Wert binär null
oder (auf BS2000-Systemen) den Wert der K-Taste (1 - 14), falls betätigt.

KCICFINF

In das Feld KCICFINF Informationen des Formatierungssystems:

- NO(NO control field)
  Die Eingabe enthält kein Steuerfeld; entweder kam die Eingabe im Zeilenmodus oder (auf BS2000-Systemen) das Format enthält kein Steuerfeld oder in ein Steuerfeld wurde nichts eingegeben.

Nur auf BS2000-Systemen:

- MO (**MO**re control fields)
  Die Eingabe enthält mehrere Steuerfelder.
- ON (**ON**e control field)
  Die Eingabe enthält genau ein Steuerfeld.

- UN **(UN** successful)
  Die Eingabe konnte nicht formatiert werden.
  Daher sind auch keine Angaben über Steuerfelder möglich. Tritt z.B. auf nach einer Eingabe in ein Format, das mit KDCOUT abgeholt worden war.

KCILTERM

In das Feld KCILTERM den Namen LTERM-Partners, über den das Terminal angeschlossen ist.

KCIUSER

In das Feld KCIUSER die aktuelle Benutzerkennung.

In die Felder des Ausgabebereichs tragen Sie ein:

KCINTAC

Wenn ein neuer Vorgang gestartet werden soll:
in das Feld KCINTAC den Transaktionscode des Teilprogramms, das den nächsten Vorgang startet.

KCINCMD

Wenn ein Benutzer-Kommando ausgeführt werden soll:
in das Feld KCINCMD das Benutzerkommando (KDC-Kommando).
Bei einer Fehlermeldung (KCICCD = ER) oder bei Fortsetzen des Vorgangs (KCICCD = CC, Folgetac wird beim PEND eingetragen) müssen Leerzeichen eingetragen werden.

KCICCD

in das Feld KCICCD je nachdem, welche Wirkung die Eingabe haben soll:

- ER (**ER**ror indication)
  für eine Fehlermeldung an das Terminal. In das Feld KCIERRCD kann dann ein Insert für diese Meldung eingetragen werden.
- CC (**C**ontinue **C**onversation)
  für die Fortsetzung des Vorgangs. Dies darf nicht nach Vorgangsende (KCICVST = EC) angegeben werden.
- SC (**S**tart new **C**onversation)
  wenn ein neuer Vorgang gestartet werden soll; nur nach Vorgangsende erlaubt (KCICVST = EC).
- ST (**ST**ack Conversation)
  wenn der aktuelle Vorgang gekellert und ein neuer Vorgang gestartet werden soll; nur erlaubt bei Transaktionsende (KCICVST = ET/RS).
- CD (process **C**omman**D**)
  wenn openUTM ein Benutzer-Kommando ausführen soll.

KCICUT

in das Feld KCICUT den Wert Y, falls der TAC bei Vorgangsbeginn abgeschnitten werden soll (nur erlaubt bei KCICCD = SC/ST), sonst N.

KCIERRCD

in das Feld KCIERRCD eine bis zu 4 Byte lange Zeichenfolge, die bei KCICCD=ER mit der Meldung K098 an das Terminal geschickt werden soll. Andernfalls (KCICCD != ER) Leerzeichen.

Zweiter Parameterbereich KCCFC/kccf.h (für BS2000-Systeme)

Mit Hilfe des zweiten Parameters übergibt openUTM die Inhalte der Steuerfelder von Bildschirmformaten an das Teilprogramm. Dieser zweite Parameter wird deshalb auch Steuerfeldbereich genannt. Im Feld KCCFS kann openUTM Eingaben, die in mehreren verschiedenen Steuerfeldern eines Formats (auch verschiedener Teilformate eines Formats) gemacht wurden, übergeben.

Die ersten beiden Felder, KCCFCREM und KCCFCFLD, entsprechen dem Steuerfeldbereich früherer openUTM-Versionen. Sie sind aus Kompatibilitätsgründen noch enthalten. Relevant sind die Felder KCCFNOCF und KCCFS.

Die folgende Tabelle zeigt den Aufbau des Steuerfeldbereichs, in dem openUTM die Eingabeparameter zur Verfügung stellt.

2. Parameterbereich (wird von openUTM versorgt)	
Feldname	**Inhalt**
KCCFCREM (Byte 1 - 8)	erste 8 Zeichen der Eingabe
KCCFCFLD (Byte 9 - 140)	Formatkennzeichen
KCCFNOCF (Byte 141 - 144)	Transaktionscode des Vorgangs
KCCFS (Byte 145 - 7744)	Tabelle (Array) der Steuerfeldinformation, jedes Array-Element enthält die Information zu einem Steuerfeld. Es gibt maximal 50 Elemente. Die Elemente sind wie folgt aufgebaut:
	• KCCFFNAM (8 Byte) Name des Formats bzw. Teilformats, das das Steuerfeld enthält
	• KCCFREM (8 Byte) Remark wie mit dem IFG definiert
	• KCFLOFL (4 Byte) Länge des Steuerfeldes
	• KCCFFLD (132 Byte) Inhalt des Steuerfeldes

Im 2. Parameterbereich stellt openUTM folgende Werte zur Verfügung:

KCCFCREM

KCCFCREM enthält den Remark, der bei der Formaterstellung mit dem IFG für das Steuerfeld definiert wurde, dessen Inhalt in KCCFCFLD enthalten ist. Wurde bei der Formaterstellung kein Remark erzeugt oder wurden in das Steuerfeld keine Eingaben gemacht, sind die 8 Byte dieses Feldes mit Leerzeichen belegt.

KCCFCFLD

KCCFCFLD enthält die Eingabe in ein Steuerfeld des Formats, sofern im Format wenigstens ein Steuerfeld definiert ist und in dieses Steuerfeld eine Eingabe gemacht wurde. KCCFCFLD enthält diese Eingabe in der Länge des Steuerfeldes. Der Rest des Feldes KCCFCFLD ist mit Leerzeichen belegt.
Erfolgte keine Eingabe in ein Steuerfeld oder besitzt das Format kein Steuerfeld, so enthalten KCCFCREM und KCCFCFLD Leerzeichen und KCCFNOCF den Wert Null. Besitzt das Format mehrere Steuerfelder, gegebenenfalls in verschiedenen Teilformaten, so beziehen sich Remark und Inhalt des Steuerfeldes auf ein Steuerfeld, in das eine Eingabe erfolgte. Erfolgte eine Eingabe in mehrere Steuerfelder, so erhält KCCFCFLD die Eingabe in das 1. Steuerfeld, in dem Teilformat, das am Bildschirm an der obersten Position steht. KCCFCREM enthält den dazugehörigen Remark.

KCCFNOCF

KCCFNOCF enthält die Anzahl der Steuerfelder des Formats, in die Eingaben gemacht wurden.

KCCFS

KCCFS enthält ein Array, das alle von FHS übergebenen Steuerfelder enthält; auch das Steuerfeld, dessen Daten bereits in den Feldern KCCFCFLD und KCCFCREM stehen. Die Anzahl der gültigen Arrayelemente steht im Feld KCCFNOCF.
Jedes Arrayelement besteht aus einer Struktur, die folgende Felder enthält:

KCCFFNAM

enthält in 8 Byte den (Teil-) Formatnamen des Formats, zu dem das Steuerfeld gehört. Ist der Name kürzer als 8 Zeichen, dann wird der Rest des Feldes mit Leerzeichen belegt.

KCCFREM

enthält den Remark, der bei der Formaterstellung mit dem IFG für dieses Steuerfeld definiert wurde. Wurde bei der Formaterstellung kein Remark erzeugt, sind die 8 Byte dieses Feldes mit Leerzeichen belegt.

KCCFLOFL

Länge des Steuerfeldes.

KCCFFLD

enthält die Eingabe in das Steuerfeld des Formats. KCCFFLD enthält diese Eingabe in der Länge des Steuerfeldes. Der Rest des Feldes KCCFFLD ist mit Leerzeichen belegt.

Eingaben in Steuerfelder (BS2000-Systeme)

Eine Eingabe in ein Steuerfeld kann unterschiedliche Ursachen haben:

- Eingabe des Terminal-Benutzers in das Steuerfeld.
- Das Feld wurde bei der Formatgenerierung mit IFG mit der Eigenschaft "automatische Eingabe" gekennzeichnet.
- Bei Formaten oder +Formaten: Das Feld hat die Feldeigenschaft "ungeschützt" und der FHS-Startparameter ISTD=RUNP (Read unprotected) ist gesetzt.

Fehler beim INPUT-Exit

Bei Fehlern im INPUT-Exit wird ein offener Vorgang nicht beendet; allerdings wird der Benutzer am Terminal mit der Meldung K098 über den Fehler informiert, falls

- der Eintrag in KCICCD (Wirkung der Eingabe) unzulässig ist oder
- der Eintrag in KCICCD nicht zu den Werten der restlichen Ausgabefelder passt.

In beiden Fällen wird zur Diagnose ein UTM-Dump mit REASON=INPERR geschrieben. Auf BS2000-Systemen wird in diesen Fällen zusätzlich ein USERDUMP geschrieben.

Datenbank-Aufrufe sind im INPUT-Exit nicht erlaubt.
Bei DB-Aufrufen wird auf BS2000-Systemen ein USERDUMP mit dem Fehlercode KDCDB10 erzeugt.

Generierungshinweise

Den Event-Exit INPUT muss man bei der Generierung mit der EXIT-Anweisung, Operand USAGE=(INPUT,...) definieren. Man kann auch mehrere INPUT-Exits für unterschiedliche Zwecke definieren. Es stehen folgende Möglichkeiten zur Verfügung:

- Eine Anwendung erhält nur einen universellen INPUT-Exit, der sowohl bei Eingaben im Formatmodus als auch bei Eingaben im Zeilenmodus aufgerufen wird. Generiert wird dieser Universal-Exit mit USAGE=(INPUT,ALL). Weitere INPUT-Exits sind in diesem Fall nicht zulässig.
- Für jede Art des Formatkennzeichens kann ein spezieller INPUT generiert werden. Es gibt mehrere Typen, jeder Typ darf höchstens einmal vorhanden sein:
  - Mit USAGE=(INPUT,LINEMODE) wird ein INPUT-Exit für Eingaben im Zeilenmodus definiert.

  Nur auf BS2000-Systemen:
  - Mit USAGE=(INPUT,FORMMODE) wird ein INPUT-Exit +Formate und *Formate definiert. Ein so generierter INPUT-Exit wird auch bei den #Formaten aufgerufen.
  - Mit USAGE=(INPUT,USERFORM) wird ein INPUT-Exit für benutzereigene Formatierungsroutinen (-Formate) definiert.

Werden in einer Anwendung spezielle INPUT-Exits verwendet, dann darf man keinen universalen Typ mit (USAGE=INPUT,ALL) definieren.

9.1.2 Event-Exit HTTP

Der Event-Exit HTTP dient dazu bestehende Teilprogramme für die Aufrufe von HTTP-Clients zu ertüchtigen, ohne dass dafür die Teilprogramme selbst geändert werden müssen.
Nachrichten von und an HTTP-Clients sind normalerweise in einem anderen Format aufgebaut, als sie von bestehenden Teilprogrammen erwartet und verarbeitet werden können. Aufgabe des HTTP-Exits ist es, die Eingabenachrichten von HTTP-Clients in die von einem Teilprogramm erwartete Datenstruktur umzusetzen und in gleicher Weise Ausgabenachrichten eines Teilprogramms so umzuformatieren, dass diese für einen HTTP-Client lesbar sind.
Um die Verarbeitungslogik eines HTTP-Exit-Programms einfach halten zu können, kann über die Generierung jedem TAC ein eigenes HTTP-Exit-Programm zugeordnet werden.

Der Event-Exit HTTP wird beim Lesen der Eingabe-Nachricht von einem sowie vor dem Senden der HTTP-Response an einen HTTP-Client aufgerufen.
Der HTTP-Exit wird für eine Eingabenachricht nur einmal, zum Zeitpunkt des ersten MGET-Aufrufs aufgerufen und muss dabei die gesamte Eingabenachricht bearbeiten und die Nachrichtenteile für den ersten und alle folgenden MGET-Aufrufe des Teilprogramms erzeugen. Dazu kann der Exit mit den `kcHttpGet`-Funktionen auf alle Bestandteile eines HTTP-Requests zugreifen z.B. mit der Funktion `kcHttpGetReqMsgBody` auf den Message Body des HTTP Requests. Mit der Funktion `kcHttpPutMgetMsg` kann die für das Teilprogramm umformatierte Nachricht bereitgestellt werden. Der Exit hat darüberhinaus die Möglichkeit mittels der Struktur `kcHttpExitMsgInfo` die Aufteilung der Eingabenachricht in Nachrichtenteile festzulegen. D.h. der Exit bestimmt damit die Anzahl und die Länge der Nachrichtenteile, die das Teilprogramm mit MGET lesen kann.
Vor dem Senden einer HTTP-Response wird der HTTP-Exit zur Bearbeitung der Ausgabenachricht aufgerufen. Der Exit hat dann die Möglichkeit mit der Funktion `kcHttpGetMputMsg` die vom Teilprogramm mit MPUT-Aufrufen an UTM übergebene(n) Nachricht(enteile) zu lesen und mit der Funktion `kcHttpPutRspMsgBody` den Message Body der HTTP-Response festzulegen. Dazu wird dem Exit in der Struktur `kcHttpExitMsgInfo` die Aufteilung der vom Teilprogramm mit MPUT-Aufrufen erzeugten Ausgabenachricht übergeben, d.h. er erhält Information über die Anzahl und die Länge der Nachrichtenteile.

Weiterhin kann der HTTP-Exit mit der Funktion `kcHttpPutStatus` die HTTP-Status Line und mit der Funktion `kcHttpPutHeader` HTTP-Header-Felder für die HTTP-Response versorgen. Im folgenden Bild ist der Ablauf skizziert.

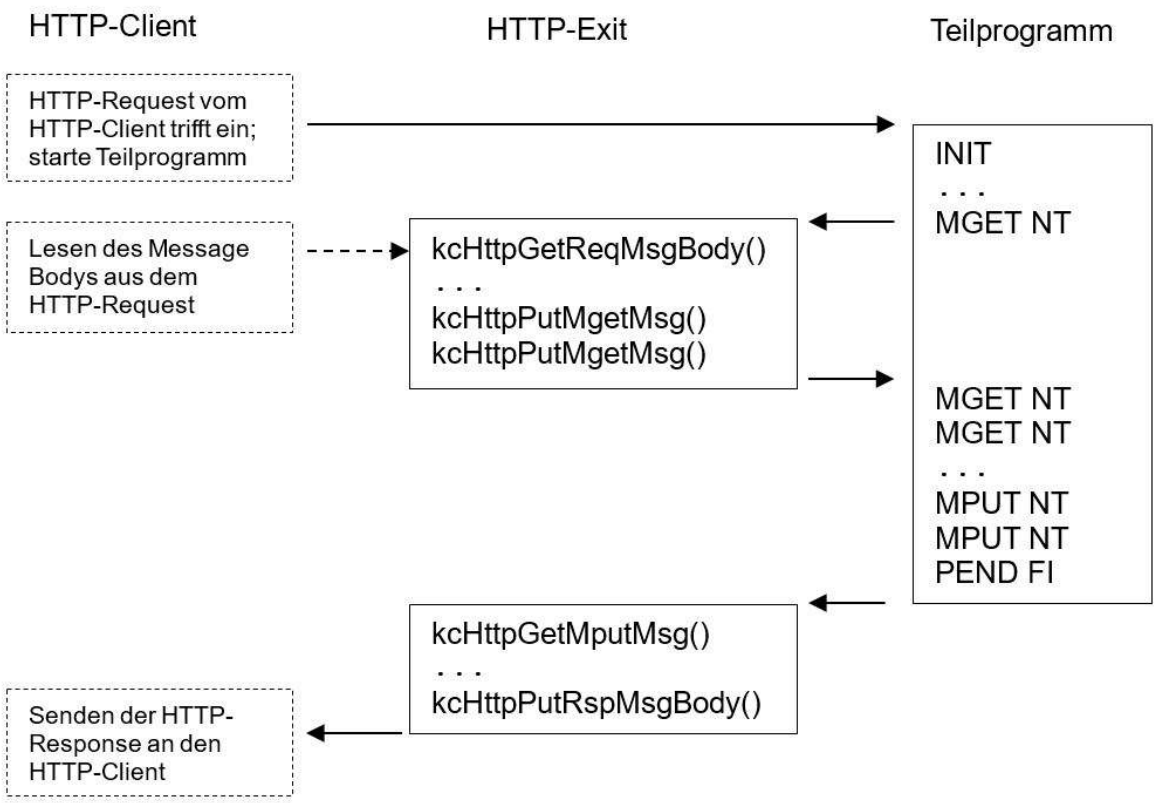


Bild: Ablauf der Nachrichtenbehandlung durch einen Event-Exit HTTP

Programmierhinweise

- Der Event-Exit HTTP muss in der Programmiersprache C implementiert werden.
- Der Event-Exit HTTP darf keine KDCS-Aufrufe verwenden.

Generierungshinweise

- Für jeden Event-Exit HTTP ist eine PROGRAM-Anweisung nötig.
- Mit der Anweisung HTTP-DESCRIPTOR wird eine Verknüpfung von der URL eines HTTP-Requests zu einem HTTP-Exit-Programm, sowie zu einem TAC der Anwendung und damit einem Teilprogramm festgelegt.

Generierungsanweisung HTTP-DESCRIPTOR

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

Prototyp des Event-Exit HTTP

```
extern void <program-name>(kcHttpExitPar *);
```

Datenstrukturen für den Event-Exit HTTP

```
typedef enum
   {  KC_UTM_HTTP_VERSION      =  1        /* current version of UTM HTTP interface    */
    , KC_UTM_HTTP_VERSION_1    =  1        /* UTM HTTP interface version of UTM V07.0A */
    , KC_UTM_HTTP_VERSION_MAX  =  INT_MAX
   } kc_http_version;

typedef struct {                           /* structure for message   */
    int             nrMsgParts;            /* number of message parts */
    short           lthMsgParts[254];      /* length of message parts */
    } kcHttpExitMsgInfo;

typedef struct {                           /* parameter structure for HTTP exit */
    kc_http_version        utmHttpVersion; /* version of UTM-HTTP interface     */
    char                   ioIndicator;    /* 'I' Input; 'O' Output             */
    unsigned char          retcode;        /* ok: 0; nok: 1-255                 */
    struct kcHttpInfo      httpInfo;       /* http info from INIT PU            */
    void *                 pCa;            /* ptr to communication area         */
    kcHttpExitMsgInfo *    pStructureInfo; /* ptr to message structure info     */
    } kcHttpExitPar;
```

Die Felder der Datenstruktur `kcHttpExitMsgInfo` haben folgende Bedeutung:

`nrMsgParts`	Anzahl der Nachrichtenteile
`lthMsgParts[254]`	Feld mit den Längen der maximal 254 Nachrichtenteilen

Die Felder der Datenstruktur `kcHttpExitPar` haben folgende Bedeutung:

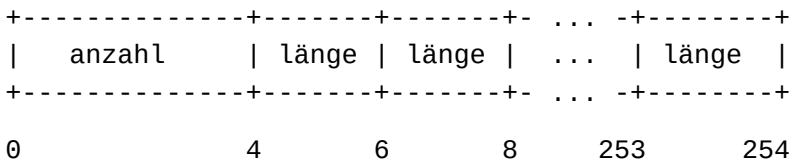
`utmHttpVersion`	Version der UTM-HTTP-Schnittstelle

`ioIndicator`	Indikator, ob der Event-Exit HTTP für die Behandlung der Eingabe-Nachricht oder der Ausgabe-Nachricht aufgerufen wurde:
	`'I' Input`
	`'O' Output`

`retcode`	In diesem Feld gibt der Event-Exit HTTP einen Returncode zurück. Hat das Feld einen Wert ungleich 0, so wird der Vorgang mit PEND ER beendet, wobei der Wert des Feldes in den sekundären KDCS-Returncode umgesetzt wird (der Wert 26 würde z.B. den sekundären KDCS-Returncode HX1A ergeben).
`httpInfo`	In dieser Struktur übergibt UTM die HTTP-Information, die einem Teilprogramm beim KDCS-Aufruf INIT PU zurückgegeben wird, an den Event-Exit HTTP.
`pCa`	Adresse des KB

pStructureInfo

Adresse der Strukturinformation der Nachricht. Beim Aufruf des Event-Exit HTTP beim Lesen der Nachricht mit MGET muss der Exit die Datenstruktur versorgen.
Beim Aufruf vor dem Senden der HTTP Response stellt UTM dem Exit in der Datenstruktur die Anzahl und Länge der vom Teilprogramm mit MPUT an UTM übergebenen Nachrichtenteile zur Verfügung.



9.1.3 Event-Exit START

START-Exits werden beim Start des Anwendungsprogramms aufgerufen, auch beim Neuladen des Programms nach PEND ER oder beim Austausch des gesamten Anwendungsprogramms.
Die Event-Exits START werden in der Reihenfolge der entsprechenden EXIT-Anweisungen aus der KDCDEF-Datei aufgerufen.

START-Exits können Sie z.B. verwenden, um beim Arbeiten mit Dateien die Dateien zu öffnen.

Programmierhinweise

- In diesem Event-Exit können Sie auf KB-Kopf und SPAB zugreifen, der KB-Programmbereich und der SPAB enthalten jedoch keine relevanten Daten. openUTM trägt für dieses Exit-Programm in die Felder KCTACVG /kccv_tac und KCTACAL/kcpr_tac des KB-Kopfes den Transaktionscode "STARTUP" ein.
- Bei Start des ersten Prozesses der Anwendung trägt openUTM in das Feld KCKNZVG/kccv_status das Vorgangs-Kennzeichen "F" ein, sonst Leerzeichen.
- Wenn Sie weitere gemeinsame Speicherbereiche (AREAs) verwenden, können Sie im START-Exit auf diese Bereiche zugreifen. Näheres siehe Abschnitte „Weitere Datenbereiche (AREAs)“ und Punkt „Erweiterung der LINKAGE SECTION" in Abschnitt „COBOL-Teilprogramm als Unterprogramm".
- Der Event-Exit START darf keine KDCS-Aufrufe verwenden.
- Der Event-Exit START wird mit einer Rücksprunganweisung verlassen.
- Erst nach dem ersten Prozess einer Anwendung werden die weiteren Prozesse (so weit vorgesehen) gestartet.
- Tritt im Event-Exit START ein Fehler auf (weil z.B. versucht wurde, eine nicht vorhandene Datei zu öffnen), so kann der Event-Exit START dafür sorgen, dass der Prozess beendet wird. Zuvor muss aber unbedingt eine Fehlernachricht geschrieben werden.
- Treten beim Startexit des ersten Prozesses der UTM-Anwendung nicht behebbare Fehler auf, die den Lauf der UTM-Anwendung verhindern, so darf die Startexitroutine nur mit *exit*(-1) beendet werden (für COBOL85: RETURN-CODE auf -1 setzen mit anschließendem STOP RUN). Es wird dann der Start der UTM-Anwendung abgebrochen.

Unix-, Linux- und Windows-Systeme

- Wird ein Startexit eines Folgeprozesses einer UTM-Anwendung mit *exit(-1)* beendet (COBOL85: RETURN-CODE auf -1 und STOP RUN), so verliert die UTM-Anwendung einen Prozess. Deshalb darf bei einem Folgeprozess der START-Exit nicht auf diese Weise beendet werden, sondern der Zustand muss von den Teilprogrammen behandelt werden, notfalls durch Herunterfahren mittels der Administrationsschnittstelle.

BS2000-Systeme

- Wenn das Anwendungsprogramm beendet und zusätzlich ein USERDUMP geschrieben werden soll, rufen Sie z. B. ein kleines Assembler-Programm auf, in dem Sie ggf. zuerst das CDUMP-Makro und zum Schluss TERM mit dem Operanden UNIT=STEP aufrufen.

Generierungshinweise

- Das Programm für den Event-Exit START muss man bei der Generierung mit der EXIT-Anweisung, Operand USAGE=START definieren.
- Pro Anwendung darf es maximal 8 Event-Exit START geben.

Mit der in dieser Version neuen Möglichkeit, mehrere START-Exits zu verwenden, können Sie besser mit vorkonfigurierten oder zugekauften Anwendungskomponenten arbeiten, da diese häufig eigene START- bzw. SHUT-Exits haben, die nun nacheinander abgearbeitet werden. Außerdem können Sie als Anwendungsbetreiber auch noch eigene START-Exits aufnehmen.

9.1.4 Event-Exit SHUT

SHUT-Exits werden bei der Beendigung des Anwendungsprogramms aufgerufen (z.B. ausgelöst durch PEND ER). SHUT-Exits können Sie z.B. verwenden, um eigene Dateien zu schließen.
Die Event-Exits SHUT werden in der Reihenfolge der entsprechenden EXIT-Anweisungen aus der KDCDEF-Datei aufgerufen

Programmierhinweise

- In diesem Event-Exit können Sie auf KB-Kopf und SPAB zugreifen, der KB-Programmbereich und der SPAB enthalten jedoch keine relevanten Daten. openUTM trägt für dieses Exit-Programm in die Felder KCTACVG /kccv_tac und KCTACAL/kcpr_tac des KB-Kopfes den Transaktionscode "SHUTDOWN" ein.
- Wenn Sie weitere gemeinsame Speicherbereiche (AREAs) verwenden, können Sie im SHUT-Exit auf diese Bereiche zugreifen. Näheres siehe Abschnitte „Weitere Datenbereiche (AREAs)“ und Punkt „Erweiterung der LINKAGE SECTION" in Abschnitt „COBOL-Teilprogramm als Unterprogramm".
- Bei normaler Beendigung der Anwendung trägt openUTM beim letzten Prozess der Anwendung in das Feld KCKNZVG/kccv_status das Vorgangs-Kennzeichen "L" ein, ansonsten ein Leerzeichen.
- Der Event-Exit SHUT darf keine KDCS-Aufrufe verwenden.
- Der Event-Exit SHUT wird mit einer Rücksprunganweisung verlassen.

BS2000-Systeme

- Tritt im SHUT-Teilprogramm ein Fehler auf (weil z.B. versucht wurde, eine nicht vorhandene Datei zu schließen), so kann der Event-Exit SHUT beendet werden. Geben Sie vorher eine Meldung aus, dass Sie selbst das Programm beenden. Wenn das Anwendungsprogramm beendet und zusätzlich ein USERDUMP geschrieben werden soll, rufen Sie z.B. ein kleines Assembler-Programm auf, in dem Sie ggf. zuerst das CDUMP-Makro und zum Schluss TERM mit dem Operanden UNIT=STEP aufrufen.

Generierungshinweise

- Die Programme für die Event-Exits SHUT muss man bei der Generierung mit der EXIT-Anweisung, Operand USAGE=SHUT definieren.
- Pro Anwendung darf es maximal 8 Event-Exits SHUT geben.

Mit der in dieser Version neuen Möglichkeit, mehrere SHUT-Exits zu verwenden, können Sie besser mit vorkonfigurierten oder zugekauften Anwendungskomponenten arbeiten, da diese häufig eigene START- und SHUT-Exits haben, die nun nacheinander abgearbeitet werden. Außerdem können Sie als Anwendungsbetreiber auch noch eigene SHUT-Exits aufnehmen.

Ein Beispiel für einen Event-Exit SHUT in C bzw. in COBOL finden Sie auf "Ergänzungen in C/C++: Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen" bzw. "Ergänzungen in COBOL: Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen".

9.1.5 Event-Exit VORGANG

Den Event-Exit VORGANG ruft openUTM beim Start und beim Beenden eines Vorgangs auf, ebenso bei fehlerhaftem Beenden und beim Wiederanlauf.

Auch wenn ein Vorgang aus mehreren Transaktionen besteht, die in verschiedenen Teilprogrammen bearbeitet werden, so wird beim Vorgangsende derselbe Event-Exit VORGANG wie bei Vorgangsbeginn aufgerufen.

Wird der Event-Exit VORGANG bei Vorgangsende aufgerufen und tritt dabei ein Fehler auf, so wird die letzte Transaktion des Vorgangs nicht mehr zurückgesetzt.

Programmierhinweise

- Der Event-Exit VORGANG darf keine KDCS-Aufrufe verwenden.
- Die längste Bearbeitungszeit des Event-Exits VORGANG sollte kleiner sein als die Zeit (in Sekunden), die maximal auf ein von einem anderen Prozess gesperrtes Betriebsmittel gewartet werden soll (KDCDEF-Anweisung MAX, Operand RESWAIT, Wert time2), da UTM während des Event-Exits VORGANG am Vorgangsende Betriebsmittel gesperrt hat.
- Der Event-Exit VORGANG wird mit Rücksprunganweisung verlassen.
- Sie können auf KB-Kopf und SPAB zugreifen, der KB-Programmbereich und der SPAB enthalten aber keine relevanten Daten.

  openUTM trägt für dieses Exit-Programm im Feld KCKNZVG/kccv_status des KB-Kopfes das Vorgangs-Kennzeichen ein. Es kann folgende Werte annehmen:

  F erster Teilprogrammlauf eines Dialog-Vorgangs

  C erster Teilprogrammlauf eines geketteten Vorgangs

  A erster Teilprogrammlauf eines Asynchron-Vorgangs

  R Wiederanlauf eines Vorgangs

  Z fehlerhaftes Ende eines Vorgangs

  E Ende eines Vorgangs

  D Beenden des Vorgangs bei Verbindungsabbau bzw. -verlust, falls für die UTM-Benutzerkennung RESTART=NO generiert ist.

  Ob das Exit-Programm in einem Dialog- oder Asynchron-Vorgang läuft, wird im Programmindikator (Feld KCPRIND) angezeigt:

  A Das Teilprogramm läuft in einem Asynchron-Vorgang

  D Das Teilprogramm läuft in einem Dialog-Vorgang

Beachten Sie, dass der Event-Exit VORGANG bei Verteilter Transaktionsverarbeitung nach Vorgangsende von einem anderen Prozess bearbeitet werden kann als das letzte Teilprogramm des Vorgangs.

Generierungshinweise

- Zu einer Anwendung kann es mehrere Event-Exit VORGANG geben.

- Welcher Event-Exit VORGANG für welchen Vorgang aufgerufen wird, legt man mit der TAC-Anweisung, Operand EXIT=Vorgangs-TAC fest. Auch die ereignisgesteuerten Vorgänge BADTACS, MSGTAC und SIGNON können einen Event-Exit VORGANG haben.
- Für jeden Event-Exit VORGANG ist eine PROGRAM-Anweisung nötig.

9.1.6 Event-Exit FORMAT (BS2000-Systeme)

Wenn Sie den Event-Exit FORMAT verwenden, verzichten Sie auf die Ausgabeunterstützung der Systemkomponenten FHS und VTSU. Sie müssen dann Ihre eigene Formatierung schreiben und ggf. auch selbst für den Bildschirmwiederanlauf sorgen. openUTM erkennt den Event-Exit FORMAT am Präfix "-" des Formatkennzeichens in KCMF/kcfn.

Der Event-Exit FORMAT muss in der EXIT-Anweisung mit dem Operanden USAGE=FORMAT generiert werden. Es ist pro Anwendung nur ein Event-Exit FORMAT erlaubt. Teilformate können Sie nicht verwenden.

> **i** Der Event-Exit FORMAT ist ein Programm, das Funktionen der Systemsoftware realisiert. Er muss daher sehr sorgfältig auscodiert sein und darf keine offenen Ausgänge, nicht auscodierte Pfade usw. enthalten, da ein Fehler zum Abbruch des Vorgangs oder sogar zum Anwendungsabsturz führen kann. In Zweifelsfällen beenden Sie den Event-Exit FORMAT mit Formatierungsfehler.

Der Event-Exit FORMAT kann in jeder von openUTM unterstützten Programmiersprache programmiert werden. Besonders geeignet sind die Programmiersprachen Assembler und C/C++. Die Beschreibung des FORMAT-Exits in diesem Abschnitt basiert auf der Programmiersprache Assembler - im Folgenden jedoch ein kurzes Beispiel für den Prototyp eines FORMAT-Exits in C:

Prototyp eines FORMAT-Exits in C (ANSI)

```
void FORMATEX( struct kc_ca  * const pKB
              ,char          * const pSPAB
              ,char          * const pFormatName
              ,char          * const pDevice
              ,char          * const pFormatArea
              ,char          * const pPhysicallyInOutArea
              ,char          * const pRestartArea
              ,char          * const pFormatControlArea
              ,char          * const pInOutIndicator
              ,char          * const pSecondaryReturnCode
             );
```

Adressleiste

openUTM stellt beim Aufruf des Event-Exits folgende Adressleiste zur Verfügung:

Reihenfolge der Wortadressen	Inhalt
1.	Adresse des KB
2.	Adresse des SPAB
3.	Adresse des Formatnamens
4.	Adresse des Terminaltyps
5.	Adresse des Formatierungs-Benutzerbereichs
6.	Adresse des physischen Ein-/Ausgabebereichs
7.	Adresse eines Wiederanlaufbereichs
8.	Adresse des Formatierungskontrollbereichs
9.	Adresse des Ein-/Ausgabe-Indikators

openUTM ruft die Formatierungsroutine auf, wenn man in den KDCS-Aufrufen MPUT, FPUT, DPUT, MGET oder FGET ein -Format verwendet.

Die Formatierung einer Dialog-Ausgabe-Nachricht (MPUT) erfolgt erst nach dem PEND- oder PGWT-Aufruf. Der Inhalt von KB und SPAB ist gleich dem des PEND- oder PGWT-Aufrufs. Die Formatierung einer asynchronen Ausgabenachricht erfolgt erst beim Senden. Der Inhalt von KB und SPAB ist in diesem Fall undefiniert, d.h. sie haben keinen Bezug mehr zum Teilprogramm, das den FPUT oder DPUT aufgerufen hat.

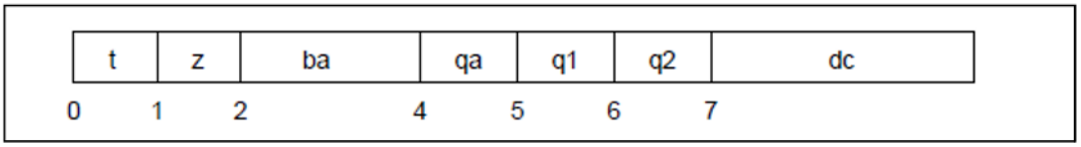
Die Formatierung einer Dialog-Eingabe-Nachricht erfolgt während der MGET-Behandlung. KB und SPAB haben den Inhalt vom MGET-Aufruf. Das gilt nur beim Event-Exit FORMAT, normalerweise formatiert openUTM vor dem INIT.
Die Formatierung einer asynchronen Eingabenachricht erfolgt beim Empfang der Nachricht, nicht erst beim FGET. Die Inhalte von KB und SPAB sind daher undefiniert.

Es ist darauf zu achten, dass vor dem Transaktionscode kein Feld sein darf, da openUTM den TAC ermittelt, ohne den Formatexit aufzurufen.

Formatname (Adresse in Wort 3)

- bei Eingabeformatierung der Formatname, der bei der vorausgehenden Ausgabeformatierung für dieses Terminal benutzt wurde (Aufrufe: MPUT, FPUT, DPUT oder Kommando KDCFOR).
- bei Ausgabeformatierung die Angabe in KCMF/kcfn des Parameterbereichs des MPUT- bzw. FPUT-Aufrufs.

Terminaltyp und Zusatzinformationen (Adresse in Wort 4)



t ist der physische Gerätetyp, wie er beim TSTAT-Makro mit TCHAR abgefragt werden kann. Die Codes lassen sich dem DCSTA-Makro entnehmen (siehe BS2000-Handbuch „Makroaufrufe an den Ablaufteil“).

Wenn der Terminaltyp vom Event-Exit FORMAT nicht unterstützt wird, **muss** ein Formatierungsfehler erzeugt werden.

z ist eine Zusatzinformation:

X’00’ = Bildschirm löschen;

X’01’ = Bildschirm nicht löschen

ba ist die Bildschirmausgabefunktion:

X’0001’: KCRESTRT

X’0001’: KCREPL

X’0002’: KCERAS

X’0004’: KCALARM

X’0008’: KCREPR

X’2000’: KCEXTEND

X’4000’: KCCARD

qa ist ein Kennzeichen, das anzeigt, ob eine Quittung angefordert werden muss - gilt nur für Ausgaben auf Drucker.

X’00’ = Quittung nicht anfordern

X’01’ = Quittung anfordern

q1, q2 Ist qa = X’01’, stehen in diesen beiden Bytes die Quittungs-Nummern im EBCDIC-Code. Man muss diese beiden Bytes bei Verwendung eines Nachrichtenkopfes in ASCII umwandeln und in die Rückmelde-Bytes RB1 und RB2 eintragen (siehe Handbücher zu Ihrem Terminal).

dc device characteristica; enthält in der Länge von 8 Byte Informationen über Gerätetyp und -ausbau, so wie dies im PDN mit XSTAT und XOPCH generiert wurde.
Die Bedeutung der 8 Byte können Sie dem DCSTA-Makro entnehmen (siehe BS2000-Handbuch „Makroaufrufe an den Ablaufteil“).

Formatierungs-Benutzerbereich (Adresse in Wort 5)

Eingabeformatierung:

Nach dem Aufruf des Formatexits erwartet openUTM hier die logische Nachricht im Eingabeformat (siehe „Nachrichtenformate“). Die maximale Länge wird bestimmt durch den generierten Wert (siehe MAX-Anweisung, Operand NB im openUTM-Handbuch „Anwendungen generieren“). Diese maximale Länge trägt openUTM vor dem Aufruf in das Längenfeld des Formatierungs-Benutzerbereichs ein.
Falls die Umsetzung der logischen in die physische Nachricht die Bereichslänge überschreitet, **muss** ein Formatierungsfehler erzeugt werden.

Ausgabeformatierung:

Der Bereich enthält die zu formatierende Nachricht, wie sie beim MPUT bzw. FPUT im Nachrichtenbereich bereitgestellt wurde. Die Nachricht ist im Eingabeformat (siehe „Nachrichtenformate“) aufgebaut. Teilformate sind nicht erlaubt.

Physischer Ein-Ausgabebereich (Adresse in Wort 6)

Eingabeformatierung:
openUTM übergibt die Nachricht, wie sie vom Terminal kommt, im Eingabeformat.

Ausgabeformatierung:
Nach dem Aufruf des Formatexits erwartet openUTM hier die formatierte Nachricht im Ausgabeformat.

Die maximale Länge ist bestimmt durch die maximal auf der Verbindung zum Client erlaubten Nachrichtenlänge. Diese maximale Länge trägt openUTM vor dem Aufruf in das Längenfeld des physischen Ein-/Ausgabebereiches ein.
Falls bei der Umsetzung der logischen in die physische Nachricht die Bereichslänge überschritten wird, **muss** ein Formatierungsfehler oder eine Ersatznachricht erzeugt werden.

Wiederanlaufbereich (Adresse in Wort 7)

Den Wiederanlaufbereich muss man benutzen, um Bildschirmformate bei Bedarf rekonstruieren zu können, was z. B. in folgenden Situationen vorkommen kann:

- Beim KDCDISP-Kommando (Anzeigen des letzten Bildschirms)
- Wenn der Terminalbenutzer einen unterbrochenen Vorgang mit seinem zuletzt eingegebenen Format fortsetzen möchte (z.B. nach einem Verbindungsverlust oder Eingabe des Kommandos KDCOFF innerhalb eines laufenden Vorgangs)
- Wenn der Bildschirm durch eine asynchrone Ausgabe zerstört wurde und wiederhergestellt werden soll.

Man muss dafür sorgen, dass der Wiederanlaufbereich immer die aktuelle logische Nachricht enthält, die bei Bedarf ausgegeben werden soll.

Will man bei einer Ausgabeformatierung zuerst den alten Bildschirm löschen, muss man den Wiederanlaufbereich neu aufbauen.

Soll bei einer Ausgabe der alte Bildschirm nicht gelöscht werden, braucht man im Wiederanlaufbereich nur die Felder zu ändern (überschreiben), die auch am Terminal modifiziert werden.

Nach Eingaben vom Terminal muss man den Wiederanlaufbereich entsprechend aktualisieren.

Er wird beim PEND in der Länge gesichert, die für den Benutzerbereich für dieses Format bei der letzten Ausgabeformatierung angegeben wurde (KCLM bei MPUT).

Beim Wiederanlauf wird eine Ausgabeformatierung durchgeführt. Dabei ist die Adresse des Formatierungsbenutzerbereichs gleich der Adresse des Wiederanlaufbereichs (d.h. es gibt nur den Wiederanlaufbereich).

Formatierungs-Kontrollbereich (Adresse in Wort 8)

Ab Adresse + 1 dieses Bereichs müssen Sie im Event-Exit FORMAT den Rückkehrcode (sedezimal) ablegen, der aussagt, ob die Formatierung erfolgreich war.

Als Eintrag kommt infrage:

X’00’ Die Ausgabeformatierung war erfolgreich.

X’xy’ Fehlermeldungen des Benutzers, die als UTM-Rückgabecode "FRxy" im Feld KCRCDC weitergegeben werden (Formatierungsfehler). Verboten sind die Einträge X’01’, X’02’, X’03’, X’04’, X’08’, X’10’ und X’99’, weil diese Rückkehrcodes für die Zusammenarbeit von openUTM und FHS reserviert sind.

Ein-Ausgabe-Indikator (Adresse in Wort 9)

Das erste Byte des Indikators enthält den Wert:

X'00': für Eingabeformatierung

X'01': für Ausgabeformatierung

X'02': für Restart

X'03': bei KDCFOR

Im Restart-Fall sind die Adressen des Formatierungs- und Wiederanlaufbereichs identisch.

Nachrichtenformate

Im Event-Exit FORMAT werden zwei unterschiedliche Nachrichtenformate verwendet (siehe vorigen Abschnitt). Diese sind nicht unbedingt auf Halbwortgrenze ausgerichtet.

```
Eingabeformat    ------------------------------------------
                 |  länge  |  b  |  b  |  (Teil-)Nachricht |
                 ------------------------------------------
                 0         2           4
Ausgabeformat    --------------------------------------------------
                 |  länge  |  b  |  b  |  b  |  (Teil-)Nachricht  |
                 --------------------------------------------------
                 0         2                 5
```

länge ist die Länge der Gesamtnachricht (binär), inklusive des Nachrichtenvorspanns, der bei der Eingabeformatierung 4 Bytes und bei der Ausgabeformatierung 5 Bytes beträgt.

b ist ein Leerzeichen (X’40’)

Informationen über den Nachrichtenaufbau von physischen Nachrichten finden Sie in den Handbüchern zu Ihrem Terminal.

Beispiel

```
FEXIT    CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         USING KB,2
         USING SPAB,3
         USING DFORMNAM,4
         USING DFORMSDE,5
         USING DADUSERA,6
         USING DAREAFMI,7
         USING DADRSRTA,8
         USING DMDCBUSE,9
         USING DFORMMOD,10
         LM    2,10,0(1)
*
*  EIN- ODER AUSGABEFORMATIERUNG, RESTART ODER KDCFOR ?
*
         CLI   FORM#IND,X'00'            00=EINGABE, 01=AUSGABE
         BE    EINFORM                   02=RESTART, 03=KDCFOR
         CLI   FORM#IND,X'01'
         BE    AUSFORM

         CLI   FORM#IND,X'02'
         BE    RESTFORM
         CLI   FORM#IND,X'03'
         BE    FORFORM
*
* SETZEN FORMATIERUNGSFEHLER: KEIN GUELTIGER OPCODE !
*
*********************************************************  K D C F O R
*
FORFORM  DS    0H
*
*      A) PHYS. NACHR. LT. FORMATNAME AUFBAUEN
*         UND MIT "STD."-WERTEN VORBESETZEN
*      B) QUITT.-ANFORDERUNG ('FORMQA') AUSWERTEN
*         UND NACHRICHTENKOPF VERSORGEN
*      C) RESTART-BER. AUFBAUEN
*      D) RET.-CODE IM FORMAT.-KONTROLLBER. SETZEN
*
END#EXIT LM    14,12,12(13)
         BR    14
*
*********************************************  EINGABE-FORMATIERUNG
*
EINFORM  CNOP   0,4
*
*      A) PHYS. NACHRICHT LT. FORMATNAME AUSWERTEN UND
*         FORMAT.-BENUTZERBER. ('DADUSERA') AUFBAUEN GEM.
*         NACHRICHTENFORMAT F. EINGABE
*      B) RESTART-BER. VERSORGEN
*      C) RET.-CODE IM FORMAT.-KOMTROLLBER. SETZEN
*
```

```
         B      END#EXIT
         EJECT
*********************************************  AUSGABE-FORMATIERUNG
*
AUSFORM  CNOP   0,4
*
*      A) AUFBAUEN DER PHYS. NACHRICHT,
*         DABEI WIRD AUSGEWERTET:
*         -  FORMAT.-BENUTZERBER.
*         -  FORMAT-NAME
*         -  STATIONSTYP UND ZUSATZINFORMATION
*            MIT
*              1) 'FORMCLMO' (BILDSCHIRM LOESCHEN ?
*                 D.H. "FORMAT-ALT" = "FORMAT-NEU" ?)
*              2) 'FORMBA' (BILDSCHIRMAUSGABE-FUNKTION)
*              3) 'FORMQA' UND GGF. 'FORMRB'
*      B) RESTARTBEREICH VERSORGEN
*      C) RET.-CODE IM FORMAT.-KONTROLLBER. SETZEN
*
         B    END#EXIT
************************************************  R E S T A R T
*
RESTFORM CNOP 0,4
*
*      A) WIEDERANLAUF-BER. AUSWERTEN
*         UND PHYS. NACHRICHT AUFBAUEN
*      B) RET.-CODE IN FORMAT.-KONTROLLBER. SETZEN
*
         B   END#EXIT
*
         EJECT
*
*  DSECTS FUER PARAMETERUEBERGABE VON UTM
*            >>>>>>>>>>>>>>>>>> NICHT VOLLSTAENDIG DEFINIERT
*
KB       KCKBA D
*
         SPACE 5
SPAB     KCPAA D
*
         EJECT
DFORMNAM DSECT
FORMNAME DS    CL8
*
DFORMSDE DSECT
FORMSDEV DS    C
FORMCLMO DS    C
FORMBA   DS    CL2
FORMQA   DS    C
FORMRB   DS    CL2
*
DADUSERA DSECT
         DS    0H
*
DAREAFMI DSECT
         DS    0H
*
DADRSRTA DSECT
         DS    0H
```

```
*
DMDCBUSE DSECT
         DS    0H
*
DFORMMOD DSECT
FORM#IND DS    X
         END
```

9.2 Anwenderspezifische Fehlerbehandlung (Unix- und Linux- Systeme)

Auf Unix- und Linux-Systemen können Sie eine anwenderspezifische Fehlerbehandlung programmieren, so dass z. B. ein Vorgang auch bei schwerwiegenden Fehlern fortgesetzt werden kann oder zusätzliche Diagnoseunterlagen erstellt werden können.

Dazu stehen Ihnen folgende Funktionen zur Verfügung:

- KCX_REG_USER_SIGNAL_HANDLER

  Registrierung der User Signal Routine, die beim Auftreten eines Signals aufgerufen wird. Die Signal Routine muss zusätzlich erstellt werden. Wenn Sie mehrere Signal Routinen registrieren, wird immer die zuletzt registrierte Routine aufgerufen.

  Die Registrierung der User Signal Routine in einem utmwork Prozess wirkt bis zur Prozessbeendigung, sofern diese nicht zuvor wieder explizit de-registriert wird.

- KCX_UN_REG_USER_SIGNAL_HANDLER

  De-Registrierung der User Signal Routine, sofern die User Signal Routine vorher registriert worden war. Diese Funktion sollte unmittelbar vor dem PEND aufgerufen werden.

- KCX_SET_RELOAD_FLAG

  Austausch des utmwork-Prozesses nach PEND RE.

  Der Aufruf dieser Funktion veranlasst den Austausch dieses utmwork-Prozesses, wenn der aktuelle Teilprogrammlauf mit PEND RE beendet wird.

  Es wird empfohlen, diese Funktion immer bei Cobol Laufzeitfehlern in der UTM-Anwendung aufzurufen, um die verzögerten oder verschleppten Auswirkungen dieses Fehlers zu eliminieren, d.h. die Dauerlauf-Fähigkeit der Anwendung sicherzustellen.

  > **i** Bei allen anderen PEND-Aufrufen (Ausnahme PEND ER) wird der utmwork-Prozess nicht ausgetauscht.

- KCX_WRITE_DUMP

  Erzeugen eines UTM-Dumps.

  Mit dieser Funktion kann ein Anwendungsprogramm einen UTM-Dump mit einem vom Anwender festgelegten Grund schreiben lassen, ohne dass der Vorgang abgebrochen wird. Es werden von UTM (analog zum PEND ER Dump) der ROOT-Bereich und die Stack-Informationen ausgegeben sowie ein Speicherabzug (Core-Dump) erzeugt.

Hinweise, wie Sie diese Funktionen programmieren müssen, finden Sie auf "Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme)" (C/C++) sowie auf "Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme)" (COBOL).

User Signal Routine

Die User Signal Routine wird beim Auftreten eines Signals im Anwendungsprogramm von openUTM anstatt der Standard-Fehlerbehandlung ausgeführt. Die User Signal Routine wird unter folgenden Bedingungen aufgerufen:

- Beim Auftreten der Signale SIGILL, SIGFPE, SIGSEGV, SIGBUS, SIGPWR und SIGABRT im Anwendungsprogramm.
- Bei SIGALRM nach Ablauf des Programm-Laufzeit-Timer im Anwendungsprogramm.

Der Einsatz einer User Signal Routine ermöglicht:

- Behandlung von Programm-Fehlern innerhalb des UTM-Vorganges.
- Ausgabe von spezifischen Fehlermeldungen an den UPIC-Client.
- Fortsetzung des UTM-Vorgangs trotz Fehlern im Anwendungsprogramm.

Die User Signal Routine ersetzt die Standard-Fehlerbehandlung von openUTM, d.h. abnormales Vorgangsende mit PENDER / XT*nn* (*nn* = Signalnummer), Ausgabe von K017 an Terminal-Clients, bei UPIC-Clients in UPIC-Logfile (+ Fehlercode "CM_DEALLOCATED_ABEND"), PEND ER Dump und Austausch des utmwork-Prozesses.

Die User Signal Routine sollte nur "Aufräumaktionen" veranlassen und danach das Programm mit PEND RE beenden. Typische "Aufräumaktionen" eines Dialog-Programms sind z.B. folgende KDCS-Aufrufe:

- RSET zum Zurücksetzen der Transaktionsdaten,
- MPUT NE mit geeigneter Fehlermeldung an den Client,
- PEND RE für die Festlegung des Folge-Programms, das nach dieser Fehlersituation laufen soll.

Es wird immer die zuletzt registrierte Signal Routine aufgerufen.

Zusätzlich wird von UTM der betroffene utmwork-Prozess nach dem nächsten KDCS-Aufruf PEND RE ausgetauscht.

Ein Beispiel finden Sie auf "User Signal Routinen (Unix- und Linux-Systeme)" (C-Programm) bzw. "User Signal Routinen (Unix- und Linux-Systeme)" (COBOL-Programm).

> **i** Bitte beachten Sie, das im Anwendungsprogramm keine Timer-Funktionen wie die C-Bibliotheksfunktion alarm() verwendet werden dürfen, zumindest dann nicht, wenn für den aktuellen TAC ein Timer generiert ist (KDCDEF-Generierung TAC RTIME= <rtime>)

9.3 STXIT-Routinen (BS2000-Systeme)

Auf BS2000-Systemen ist es möglich, für bestimmte Ereignisse (z.B. Adressfehler, Programmende) STXIT-Routinen zu definieren. Diese Routinen werden vom Betriebssystem (nicht von openUTM) aktiviert, wenn ein solches Ereignis eintritt (siehe BS2000-Handbuch „Makroaufrufe an den Ablaufteil“).

openUTM stellt eigene STXIT-Routinen zur Verfügung, die vor dem Aufruf des START-Exits geöffnet werden. Ausnahme: TIMER/RTIMER, die direkt nach dem START-Exit geöffnet und von openUTM für die Überprüfung der Teilprogramm-Laufzeit verwendet werden (siehe TIME-Parameter der TAC-Anweisung).

Sie können eigene, parallele STXIT-Routinen definieren, die zusätzlich zu den von openUTM definierten STXIT-Routinen aktiviert werden. Wenn Sie den Startparameter STXIT=OFF angeben, werden nur die von Ihnen erstellten STXIT-Routinen aktiviert (UTM-STXIT-Routinen werden nicht aktiviert). Letzteres ist nur möglich, wenn die Anwendung im Dialog gestartet wurde.

Die von Ihnen erstellten STXIT-Routinen werden immer vor der UTM-STXIT-Routine aktiviert (Ausnahme: RUNOUT).

¡ Falls nach der Startphase dynamisch Programme mit zugehörigem Laufzeitsystem nachgeladen werden, die eigene STXIT-Routinen anmelden, kann die Aktivierungsreihenfolge der STXIT-Routinen von der hier beschriebenen abweichen.

Die von Ihnen erstellten STXIT-Routinen müssen mit EXIT CONTINU=YES beendet werden. Andernfalls garantiert openUTM nicht für eine korrekte Fehlerbehandlung (z.B. PEND ER in bestimmten Situationen). Die STXIT-Routinen von openUTM werden mit EXIT CONTINU=NO beendet.

¡ Für STXIT-Behandlung in ILCS-Programmen beachten Sie bitte auch die Beschreibung in Abschnitt „Ereignisbehandlung in ILCS-Programmen (BS2000-Systeme)“.

¡ Bitte beachten Sie, das im Anwendungsprogramm keine Timer-Funktionen wie die C-Bibliotheksfunktion alarm() oder das Makro SETIC (Intervallzeitgeber setzen) verwendet werden dürfen, zumindest dann nicht, wenn für den aktuellen TAC der jeweilige Timer generiert ist (KDCDEF-Generierung TAC TIME= {time1 | (time1,time2)}).

9.4 Ereignisbehandlung in ILCS-Programmen (BS2000- Systeme)

In einer ILCS Anwendung werden alle Unterbrechungen zuerst an die Ereignisroutinen von ILCS geleitet. Ob und wie ein Ereignis behandelt wird hängt ab:

- vom Ereignis
- von der Programmiersprache, in der das zuletzt aktive Programm geschrieben ist
- von den Ereignisbehandlungsroutinen der beteiligten Programme
- von der Aufrufsequenz, sofern in dieser verschiedene ILCS Programmiersprachen vorkommen

Varianten der Ereignisbehandlung:

1. Das Teilprogramm (d.h. ein Programm innerhalb der Call-Aufrufsequenz) oder ein in dessen Aufrufsequenz beteiligtes Laufzeitsystem behandelt das Ereignis und setzt die Verarbeitung fort.
2. Im obigen Rahmen des Teilprogramms wird das Ereignis nicht behandelt, es wird mit Context und Ereigniscode zu openUTM durchgereicht.

   Hier sind zwei Fälle zu unterscheiden:

   a. Es handelt sich um ein Ereignis der Klasse PROCHK oder ERROR:

      openUTM bereitet die Meldung K102 auf mit dem übergebenen Context und mit dem übergebenen IW.Das IW ist hier zusätzlich Steuerelement für die folgende PEND Behandlung.

   b. Es handelt sich um ein OTHER EVENT:

      Beim Ereignis ’Other Event’ kann openUTM nur stereotyp reagieren und muss dabei verhindern, dass die Task abnormal beendet wird. Der übergebene Ereigniscode ist kein IW im STXIT Sinne; sein Wert hat für openUTM keine Bedeutung; es muss insbesondere verhindert werden, dass dieser Ereigniscode in seiner Eigenschaft als Steuerelement der UTM-PEND-Behandlung zu Fehlerinterpretationen führt. Damit die UTM-Fehlerbehandlung richtig ablaufen kann, wird deshalb der Ereigniscode auf den Wert X’FF’ gesetzt. Dieser Wert wird im weiteren Ablauf von openUTM als (simuliertes) IW interpretiert. D.h. wird ein ’Other event’ bis zu openUTM durchgereicht, so verhält sich openUTM so als wäre ein STXIT-Ereignis mit dem IW=X’FF’ eingetreten.

      Die Diagnose muss sich in diesem Fall auf die Meldung stützen, die beim Auftreten eines ’Other Event’ von der beteiligten Sprache, dem Teilprogramm oder von ILCS ausgegeben wird. Fehlt eine solche Meldung, dann wird mit der UTM-Meldung K102...KDCIWFF das STXIT-Ereignis X’FF’ angezeigt.
      Der dabei entstehende User-Dump enthält den Code KDCIWFF.
3. Im obigen Rahmen des Teilprogramms wird das Ereignis interpretiert, evtl. eine Meldung ausgegeben und der ILCS Entry IT0TERM aufgerufen. Über diesen Entry wird openUTM aktiviert und simuliert die Ereignisbehandlung eines TERM UNIT=PROG. In diesem Fall können die Ereignisdaten der primären Unterbrechung nur der Meldung des Teilprogramms oder der beteiligten Sprache entnommen werden. openUTM gibt die Meldung K102... KDCIW90 (TERM) aus, mit den Registerständen zum Zeitpunkt des IT0TERM Aufrufs.
   Ein entstehender User-Dump enthält den Code KDCIW90.
4. Es tritt ein ILCS interner Fehler auf:

   ILCS verzweigt auf einen definierten UTM-Entry; openUTM setzt einen internen Ereigniscode IW=X’00’, der zweierlei bedeuten kann:

   - Es trat ein ILCS interner Fehler auf
   - die ILCS-Stack-Kette ist unterbrochen

UTM gibt dieses IW=X’00’ in der Meldung K102 aus. In seiner Funktion als Steuerelement wird dieses IW=X’00’ im weiteren Programmablauf von openUTM wie ein IW=X’88’ interpretiert, d.h. die Task wird mit TERM UNIT=STEP beendet.

STXIT-Ereignisse - Verhalten von ILCS im Fehlerfall

Nach STXIT-Ereignissen, die ILCS bzw. die aufgerufenen EHL-Routinen nicht bearbeiten kann, schließt ILCS seine STXIT und beendet die Routine mit EXIT CONTINUE=NO. In den meisten Fällen tritt der Fehler direkt danach nochmal auf und weitere STXIT-Routinen (z.B. von openUTM) können den Fehler behandeln. In Einzelfällen, wie z. B. Exponentenüberlauf, ist die Programmumgebung (Registerstände usw.) zu Beginn der ILCS-STXIT-Routine schon so verändert, dass ein Wiederherstellen der Fehlerumgebung nicht möglich ist. ILCS prüft vor dem Beenden der STXIT anhand des IW, ob der aufgetretene Fehler wieder auftreten kann. Ist das nicht der Fall, ruft ILCS seine Programmbeendigungsroutine IT0TERM auf.

Benutzereigene STXIT-Routinen in ILCS-Umgebung:

Um die korrekte Reihenfolge der STXIT-Ausführung in einer ILCS-Programm-Umgebung zu gewährleisten, sollten Sie benutzereigene STXITs über spezielle Assembler-Makros bzw. C-Funktionen direkt bei ILCS anmelden. Details finden Sie im jeweiligen Benutzerhandbuch zu CRTE, ASSEMH und C.

9.5 Event-Services

Event-Services sind Dialog- bzw. Asynchron-Vorgänge, die auf Grund eines bestimmten Ereignisses gestartet werden, also ereignisgesteuerte Vorgänge. Die Teilprogramme dieser Vorgänge müssen KDCS-Aufrufe enthalten und die im Kapitel „Aufbau und Einsatz von UTM-Programmen“ dargestellten Regeln einhalten.

Event-Services erhalten per Generierung einen privilegierten Transaktionscode, den openUTM intern verwendet. Es sind folgende ereignisgesteuerten Vorgänge möglich:

BADTACS mit dem TAC **KDCBADTC**
MSGTAC mit dem TAC **KDCMSGTC**
SIGNON mit dem TAC **KDCSGNTC** oder Angabe als **SIGNON-TAC** in der Anweisung BCAMAPPL

Die Transaktionscodes zu diesen ereignisgesteuerten Vorgängen werden mit der KDCDEF-Anweisung TAC definiert.

Datenbank-Aufrufe sind erlaubt, im EVENT-Service SIGNON allerdings nur, wenn dies per Generierung ausdrücklich zugelassen wurde (SIGNON ...,RESTRICTED=NO).

9.5.1 Dialog-Vorgang BADTACS

Der Dialog-Vorgang BADTACS wird - falls generiert - gestartet, wenn ein Terminal oder eine Transportsystem-Anwendung einen ungültigen Transaktionscode eingegeben hat. Das kann folgende Ursachen haben:

- der Transaktionscode ist nicht generiert.
- der Transaktionscode ist keinem Teilprogramm zugeordnet.
- Der Benutzer hat nicht die Berechtigung den Transaktionscode aufzurufen. Dies kann verschiedene Gründe haben, z.B.:
  - der Transaktionscode ist ein Administrator-TAC, aber der Benutzer hat sich nicht als Administrator angemeldet,
  - dem Transaktionscode ist ein Lockcode zugeordnet, aber der Benutzer oder der LTERM-Partner verfügt über keinen entsprechenden Keycode.

Programmierhinweise

- Der Dialog-Vorgang BADTACS muss gemäß den in Kapitel Aufbau und Einsatz von UTM-Programmen dargestellten Regeln einen MPUT-Aufruf absetzen. Dieser MPUT-Aufruf ersetzt dann die Fehlermeldung K009, die openUTM ohne BADTACS erzeugen würde.
- Bei Vorgangs-Beginn steht nach dem INIT-Aufruf der ungültige TAC im KB-Kopf, und zwar sowohl im Feld KCTACAL/kcpr_tac als auch im Feld KCTACVG/kccv_tac.
- Wird der Vorgang BADTACS aus dem Zeilenmodus aufgerufen, dann erhält er die volle Nachricht einschließlich der ersten 8 Byte. Bei Aufruf aus dem Formatmodus werden die ersten 8 (bei *Formaten) bzw. 10 (bei +Formaten) Byte aus der Nachricht entfernt.
- Außerhalb eines Vorgangs kann man BADTACS auch mittels einer Funktions-Taste starten, wenn für diese ein Returncode (20Z-39Z), aber kein TAC generiert ist (SFUNC-Anweisung, Operand RET=) oder wenn die Funktions-Taste nicht generiert wurde (Returncode 19Z). Mit dem ersten MGET wird dann der Returncode übergeben, mit einem zweiten MGET kann eine Nachricht (einschließlich der ersten 8 Byte) gelesen werden. In die Felder KCTACAL/kcpr_tac und KCTACVG/kccv_tac wird nichts eingetragen.

Generierungshinweise

- Pro Anwendung darf es nur einen Vorgang BADTACS geben,
- Das erste Teilprogramm des Dialog-Vorgangs BADTACS muss bei der Generierung mit der TAC-Anweisung TAC KDCBADTC, PROGRAM=... definiert werden.

Ein Beispiel für einen EVENT-Service BADTACS finden Sie in C auf "Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen" bzw. in COBOL auf "Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen".

9.5.2 Asynchron-Vorgang MSGTAC

Das Asynchron-Teilprogramm MSGTAC wird aufgerufen, wenn

- openUTM eine K*nnn*- oder P*nnn*-Meldung ausgibt und
- für diese Meldung als Meldungsziel MSGTAC eingetragen ist.

Wie MSGTAC als Meldungsziel eingetragen wird und wie übergebene Meldungen aufgebaut sind, ist im openUTM-Handbuch „Meldungen, Test und Diagnose“ beschrieben.

Der Vorgang MSGTAC wird von openUTM mit der Administrationsberechtigung und allen Keys der Anwendung ausgestattet, also mit dem Maximum an Berechtigungen.

Wird das MSGTAC-Teilprogramm mit KCRCCC >= 70Z abgebrochen, so wird es von openUTM für den weiteren Anwendungslauf gesperrt (STATUS=OFF). Es muss dann explizit vom Administrator freigegeben werden (STATUS=ON). Dies gilt nicht für einen Programmabbruch wegen programmiertem PEND ER/FR.

openUTM schreibt die Meldungen für das Meldungsziel MSGTAC auf den Pagepool. Solange der Warnlevel 2 des Pagepools nicht überschritten wird, ist sichergestellt, dass keine dieser Meldungen verloren geht.
Ausnahme: Die Meldungen, die das Über- bzw. Unterschreiten des Warnlevels anzeigen, können nicht in allen Fällen dem MSGTAC-Teilprogramm zugestellt werden.

Programmierhinweise

- Das Asynchron-Programm MSGTAC liest die Meldung mit einem FGET-Aufruf in den Nachrichtenbereich des Teilprogramms. Die FGET-Aufrufe sollten dabei so lange wiederholt werden, bis der Returncode 10Z gesetzt wurde, so dass alle anstehenden Meldungen in einem Teilprogrammlauf gelesen werden.
- Für jede UTM-Meldung steht eine Datenstruktur zur Verfügung, die im Teilprogramm zur Interpretation des Nachrichteninhaltes verwendet werden kann, für C/C++ sind diese in der Include-Datei *kcmsg.h* enthalten, für COBOL im COPY-Element KCMSGC.
- Das MSGTAC-Teilprogramm läuft als Asynchron-Vorgang unter der internen UTM-Benutzerkennung KDCMSGUS mit KSET=MASTER und PERMIT=ADMIN.
- Der MSGTAC-Vorgang darf nur aus einem Teilprogrammlauf bestehen.
- In einem Teilprogrammlauf muss mindestens eine Meldung mit FGET gelesen werden, da das Teilprogramm sonst abnormal beendet wird.
- Der Vorgang wird intern mit dem Administrator-TAC KDCMSGTC gestartet.

  openUTM versorgt den KB-Kopf wie folgt:

Felder im KB-Kopf		Einträge bei MSGTAC
COBOL	**C/C++**	
KCBENID	kcuserid	KDCMSGUS
KCTACVG	kccv_tac	KDCMSGTC
KCTACAL	kcpr_tac	KDCMSGTC
KCLOGTER	kclogter	KDCMSGLT
KCTERMN	kctermn	MT

- Das MSGTAC-Teilprogramm kann die Programmschnittstelle zur Administration verwenden und es kann die Administrationskommandos absetzen. Insbesondere darf das MSGTAC-Teilprogramm die KDCS-Aufrufe DADM und PADM zur Administration von DPUT-Nachrichten und Druckern absetzen, siehe openUTM-Handbuch „Anwendungen administrieren“.

Generierungshinweise

- Es darf pro Anwendung nur ein MSGTAC-Teilprogramm geben.
- Das MSGTAC-Teilprogramm muss in der TAC-Anweisung definiert werden mit

  ```
  TAC KDCMSGTC,PROGRAM=... .
  ```

Beispiel für ein MSGTAC-Teilprogramm

Das MSGTAC-Teilprogramm NOHACK soll nicht berechtigte Benutzer daran hindern, sich an eine UTM-Anwendung anzuschließen. Wenn mehr als 3-mal über einen LTERM-Partner ein Anmelde-Versuch mit ungültiger Benutzerkennung, falschem Passwort oder falschem Ausweis versucht wird, soll die Verbindung zum Terminal abgebrochen werden. Dazu müssen Sie zusätzliche Vorbereitungen treffen (siehe auch openUTM-Handbuch „Meldungen, Test und Diagnose“).

- Vorbereitungen:
  1. UTM-Tool KDCMMOD aufrufen.
  2. GEN-Kommando absetzen; dabei ist der Name des Meldungsmoduls anzugeben.
  3. Mittels MODMSG-Kommandos für die Meldungen K008, K033 und K094 als zusätzliches Meldungsziel MSGTAC definieren.
  4. Das bisher entstandene Quellprogramm übersetzen und zusammen mit der Anwendung binden.
  5. In der KDCDEF-Anweisung MESSAGE das Meldungsmodul definieren.
  6. KDCPTRMA in TAC-Anweisung definieren.

  Elegantere Lösung: Sie können die Angaben von Punkt 2. und 3. in eine Datei schreiben und diese dann als Eingabedatei für Punkt 1. verwenden.
- Realisierung des MSGTAC-Teilprogramms:

  Das MSGTAC-Teilprogramm NOHACK zählt die Anzahl der Fehlversuche in einem TLS. Akzeptiert openUTM die Anmeldung an die Anwendung (Meldung K008 oder K033), so wird dieser TLS wieder gelöscht.
  Falls nach drei ungültigen Anmeldeversuchen der 4. Versuch wieder fehlerhaft ist, so soll das entsprechende Terminal über "asynchrone Administration" abgemeldet werden. Dies geschieht mit einem FPUT-Aufruf mit KCRN = "KDCPTRMA" und einem Nachrichtenbereich mit dem Inhalt PTERM=pterm, ACT=DIS (siehe auch openUTM-Handbuch „Anwendungen administrieren“).
  Das Administrationskommando wird dann mit LPUT in der Benutzer-Protokolldatei protokolliert und der TLS gelöscht. Ein Beispiel in C finden Sie in Abschnitt "Beispiel: Event-Service MSGTAC" bzw. in COBOL in Abschnitt "Beispiel für ein Asynchron-Teilprogramm MSGTAC".

  Die K-Meldungen werden jeweils mit FGET vom MSGTAC-Teilprogramm gelesen. Nach der "Verarbeitung" einer K-Meldung wird mit FGET sofort die nächste K-Meldung gelesen, innerhalb desselben Teilprogrammlaufs. Eine Liste der K-Meldungen finden Sie im openUTM-Handbuch „Meldungen, Test und Diagnose“.

9.5.3 Anmelde-Vorgang SIGNON

Der Anmelde-Vorgang SIGNON ist ein Dialog-Vorgang, der gestartet wird

- nach dem Verbindungsaufbau eines Terminals oder einer Transportsystem-Anwendung über einen Transportsystem-Zugriffspunkt, falls für diesen Zugriffspunkt ein Anmelde-Vorgang generiert wurde, oder
- vor dem Start jeder von einem UPIC-Client initiierten Conversation, falls für den Transportsystem-Zugriffspunkt ein Anmelde-Vorgang generiert wurde und falls
  - Anmelde-Vorgänge per Generierung auch für UPIC-Clients freigeschaltet sind,
  - die Anwendung mit OMIT-UPIC-SIGNOFF=NO generiert ist
  - und der Benutzer noch nicht angemeldet ist.

Für jeden Transportsystem-Zugriffspunkt einer UTM-Anwendung kann jeweils ein eigener Anmelde-Vorgang generiert werden, siehe Operand SIGNON-TAC der Anweisung BCAMAPPL und Anweisung TAC mit dem Transaktionscode KDCSGNTC im openUTM-Handbuch „Anwendungen generieren“.

Generierungshinweise

- Der Anmelde-Vorgang für einen mit MAX APPLINAME definierten Transportsystem-Zugriffspunkt wird generiert mit `TAC KDCSGNTC,PROGRAM=...` Dieser Anmelde-Vorgang ist dann gleichzeitig Standard für alle Transportsystem-Zugriffspunkte der Anwendung.
- Mit dem Parameter SIGNON-TAC der BCAMAPPL-Anweisung kann für einen Transportsystem-Zugriffspunkt auch ein anderer Anmelde-Vorgang generiert werden. Soll für einen Transportsystem-Zugriffspunkt kein Anmelde-Vorgang verwendet werden, so muss der Wert *NONE für den Parameter SIGNON-TAC angegeben werden
- Für UPIC-Clients werden Anmelde-Vorgänge freigeschaltet durch `SIGNON ...,UPIC=YES.`
- In der SIGNON-Anweisung der Generierung kann festgelegt werden, dass sich ein Benutzer auch dann vorläufig bei der UTM-Anwendung anmelden kann, wenn die Gültigkeitsdauer seines Passworts überschritten ist (Parameter GRACE = YES). In diesem Fall muss das Passwort des Benutzers geändert werden, wenn der Anmelde-Vorgang erfolgreich beendet werden soll.

  Die maximale Anzahl von erfolglosen Anmeldeversuchen von einem Client aus lässt sich kontrollieren durch: `SIGNON ,SILENT-ALARM=` *anzahl*

  Wird dieser Wert erreicht, dann wird die Meldung K094 nach SYSLOG ausgegeben. Diese Meldung kann auch von einem MSGTAC-Programm verarbeitet werden, siehe "Asynchron-Vorgang MSGTAC".

Eine genaue Beschreibung finden Sie im openUTM-Handbuch „Anwendungen generieren“.

9.5.3.1 Programmierhinweise

Der SIGNON-Vorgang steuert das Anmeldeverfahren per Programm. Der Anmelde-Vorgang wird vor allem durch folgende KDCS-Aufrufe gesteuert, siehe "PEND Beenden eines Teilprogramms" und "SIGN An- und Abmelden steuern, Berechtigungsdaten überprüfen, Passwort ändern":

SIGN ST	Stand des Anmelde-Vorgangs abfragen. Der Aufruf liefert auch das Ergebnis eines vorangegangenen SIGN ON.
SIGN ON	Berechtigungsdaten zur Prüfung an openUTM übergeben. Der Aufruf liefert als Ergebnis nur zurück, ob der Aufruf syntaktisch korrekt war, nicht aber, ob die Anmeldung erfolgreich war.
PEND PS	Teilprogramm beenden, um von openUTM die Berechtigungsdaten prüfen zu lassen und ggf. den Zwischendialog einzuschieben. Anschließend wird der Anmelde-Vorgang im Folge-Teilprogramm fortgesetzt. Erst mit diesem Aufruf findet eine vorläufige Anmeldung statt, wenn die beim SIGN ON übergebenen Daten korrekt waren.

Mit openUTM auf BS2000-Systemen wird ein entsprechendes Beispiel-Programm ausgeliefert. Das openUTM-Handbuch „Einsatz von UTM-Anwendungen auf BS2000-Systemen“ enthält eine Beschreibung dieses Beispielprogramms sowie weitere Informationen zum Konzept des Anmelde-Vorgangs und typischen Anwendungsfällen.

Hinweise

- Im ersten Teil des Anmeldeverfahrens enthält das Feld KCBENID/kcuserid im KB-Kopf nur Leerzeichen.
- Falls die Anmeldung vor Ausführung des PEND FI noch nicht erfolgreich war, baut openUTM die Verbindung zum Terminal oder zur TS-Anwendung ab, oder beendet die UPIC-Conversation. Damit kann man mehrmaliges erfolgloses Anmelden, z.B. wegen fehlender Berechtigung, auf einfache Weise durch PEND FI nach KCRSIGN1 = U behandeln.
- Wenn der Anmelde-Vorgang die für ihn geltenden Regeln verletzt, bricht openUTM ihn mit PEND ER ab. Die Verbindung zum Terminal oder zur TS-Anwendung wird abgebaut, die Verbindung zum UPIC-Client bleibt bestehen.
- Ist die UTM-Anwendung ohne echte Benutzerkennungen generiert (d.h. ohne USER), kann sich der Anmelde-Vorgang sofort beenden, da die Anmeldung erfolgreich ist. Die entsprechende Information erhält der Anmelde-Vorgang beim Aufruf SIGN ST.

  Im Anmelde-Vorgang kann aber auch eine anwendungseigene Berechtigungsprüfung durchgeführt werden, zum Beispiel anhand einer Datenbank mit Berechtigungsdaten.
- Einschränkungen im Anmelde-Vorgang
  - Die KDCS-Aufrufe PEND RE/RS/SP sind verboten.
  - Vor erfolgreicher Anmeldung sind weder FPUT/DPUT-Aufrufe noch Zugriffe auf einen ULS erlaubt. Datenbank-Aufrufe und Zugriffe auf GSSBs und TLSs sind vor erfolgreicher Anmeldung nur dann zulässig, wenn es die Anweisung SIGNON ausdrücklich erlaubt. Daher sollte man vor den oben genannten Zugriffen erst mit SIGN ST abfragen, ob der Anmeldestatus (Rückgabe in KCRSIGN1) den Wert A bzw. R besitzt.
  - Aufrufe für verteilte Verarbeitung dürfen nicht abgesetzt werden.

9.5.3.2 Anmelde-Vorgang für Terminals

Der Anmelde-Vorgang für Terminals besteht aus zwei Teilen:

1. Teil: Berechtigungsdaten vom Terminal lesen und an openUTM übergeben.
2. Teil: Bei korrekter Anmeldung eine Bestätigung (Meldung und z.B. USER-spezifisches Startformat für das Terminal) schicken.

Zwischen erstem und zweitem Teil führt openUTM u.U. einen Zwischendialog mit dem Terminal, um weitere Berechtigungsdaten wie Ausweisinformation oder Passwort abzufragen.

Bei SIGN ST gibt openUTM im Feld KCRSIGN1 den Status des Anmeldeverfahrens zurück. Aufgrund dieser Statusanzeige entscheidet das Teilprogramm, wie es weiter vorgeht. Es gibt folgende Möglichkeiten:

- KCRSIGN1 = C (connected)
  Das Terminal ist mit der Anwendung verbunden, aber es ist noch kein Benutzer angemeldet.
  Der Benutzer am Terminal wird dann per MPUT-Aufruf aufgefordert, die Berechtigungsdaten einzugeben.
  Das Teilprogramm beendet sich mit PEND KP. Das Folgeteilprogramm liest die Berechtigungsdaten mit MGET, übergibt sie mit SIGN ON an openUTM und beendet sich mit PEND PS.
- KCRSIGN1 = I (incomplete)
  openUTM kennt bereits die Benutzerkennung, benötigt aber noch Zusatzinformationen (Passwort, Ausweis), die per Zwischendialog angefordert werden. Das Teilprogramm muss sich mit PEND PS beenden, danach führt openUTM den Zwischendialog durch.
- KCRSIGN1 = A (accepted)
  Die Anmeldung ist korrekt. Im KB-Kopf ist die Benutzerkennung eingetragen. Wenn gewünscht, können vor Vorgangsende weitere Dialog-Schritte eingeschoben werden. Der Anmelde-Vorgang wird mit PEND FI oder PEND FC beendet. Die abschließende Nachricht erzeugt der Vorgang selbst und gibt sie mit MPUT aus. Wenn sie an ein Terminal gerichtet ist, kann sie aus dem USER-spezifischen Startformat bestehen. Den Namen des Startformats liefert openUTM beim Aufruf SIGN ST im Feld KCRMF/kcrfn zurück.
  Bei MPUT PM mit anschließendem PEND FI wird, sofern vorhanden, die letzte Dialog-Nachricht des letzten Vorgangs ausgegeben.
- KCRSIGN1 = R (restart)
  Die Anmeldung ist korrekt, es liegt ein Vorgangs-Wiederanlauf vor. Im KB-Kopf ist die Benutzerkennung eingetragen.
  Wenn gewünscht, können vor Vorgangsende weitere Dialog-Schritte eingeschoben werden. Der Anmelde-Vorgang muss sich mit PEND FI beenden. Der Vorgangs-Wiederanlauf wird durch den Aufruf MPUT PM, KCLM=0, KCMF/kcfn=Leerzeichen ausgelöst. In diesem Fall gibt openUTM die gesicherte letzte Nachricht des unterbrochenen Vorgangs aus (Bildschirmwiederanlauf am Terminal) oder startet bei lokalem Sicherungspunkt nach PEND SP/ FC das Folge-Teilprogramm bzw. den Folge-Vorgang. Durch Beendigung ohne MPUT-Aufruf kann der offene Vorgang abnormal beendet werden. An ein Terminal wird eine K017-Meldung gesendet.
- KCRSIGN1 = U (unsuccessful)
  openUTM hat die Berechtigungsdaten nicht akzeptiert. Ein Terminal-Anmelde-Vorgang befindet sich noch im 1. Teil und muss den Benutzer auffordern, die Berechtigungsdaten erneut einzugeben. Beendet sich der Anmelde-Vorgang in diesem Zustand, so wird die Verbindung zum Terminal abgebaut.

LTERM-Partner mit automatischem KDCSIGN

Der Anmelde-Vorgang erhält beim Aufruf SIGN ST die Information, dass die Benutzerkennung bereits bekannt ist. Abhängig von der Generierung kann jetzt ein Zwischendialog zum Anfordern einer Magnetstreifenkarte oder eines Passworts durchgeführt werden.

Anmeldung über Multiplex-Verbindung (BS2000-Systeme)

Abhängig von der Konfiguration der Multiplex-Verbindung bei OMNIS, leitet OMNIS die Berechtigungsdaten über das Protokoll PUTMMUX zur Prüfung an openUTM weiter. Sind die Daten korrekt, startet openUTM den Anmelde-Vorgang, der mit dem Aufruf SIGN ST die entsprechende Information erhält. Sind die Daten nicht korrekt, wird der Anmelde-Vorgang nicht gestartet, und der Verteiler muss seinerseits den Benutzer am Terminal informieren.

Die folgende Abbildung zeigt den Ablauf eines Anmelde-Vorgangs am Terminal:

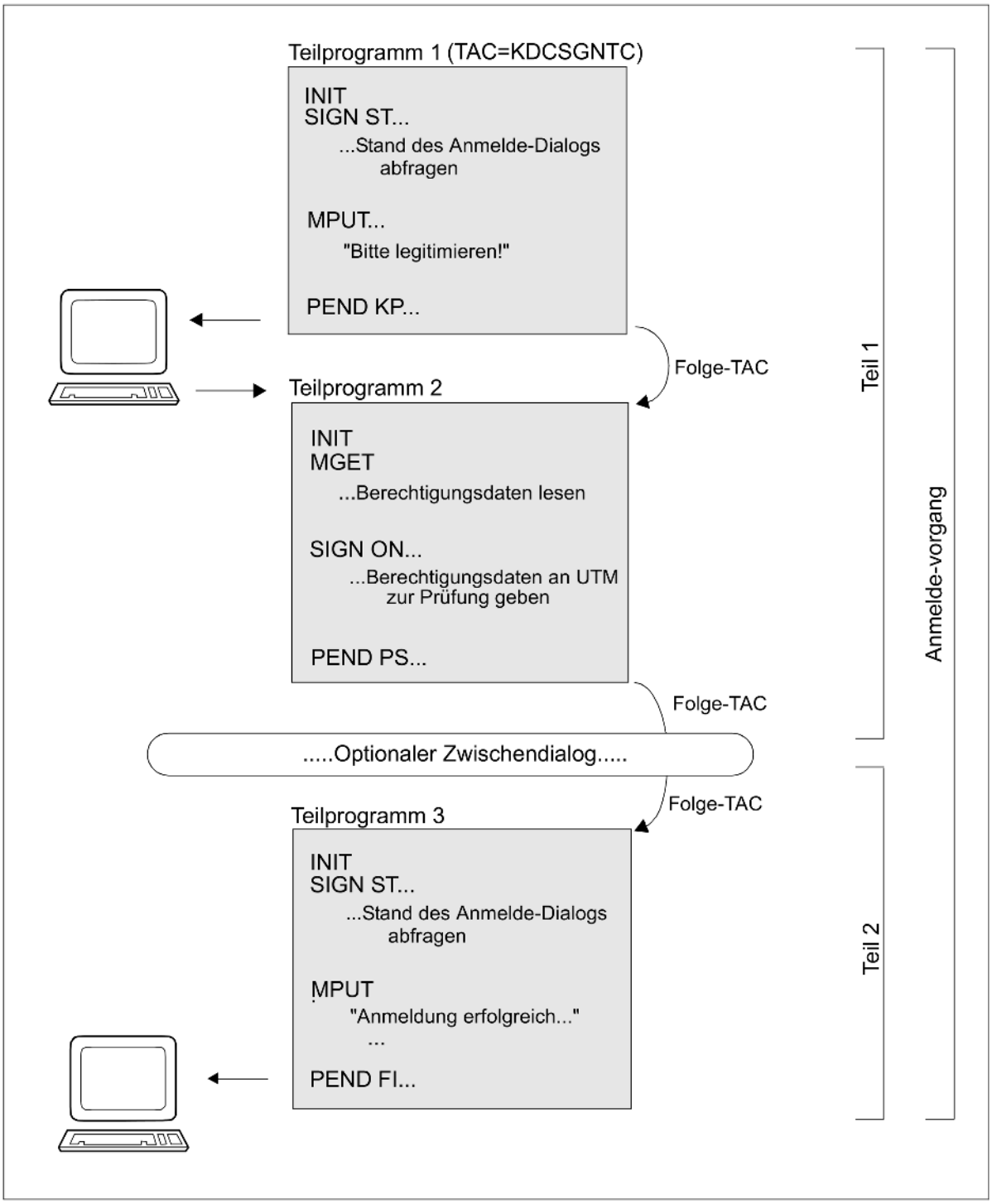


Bild: Ablauf eines Anmelde-Vorgangs für Terminals

9.5.3.3 Anmelde-Vorgang für UPIC-Clients und Transportsystem-Anwendungen

Läuft der Anmelde-Vorgang für einen UPIC-Client oder einer Transportsystem-Anwendung, befindet sich der Anmelde-Vorgang **nie im ersten Teil**, da als Benutzerkennung mindestens die Verbindungs-Benutzerkennung zugeordnet ist. Es gilt:

- Bei TS-Anwendungen wird der Benutzer unter der Verbindungs-Benutzerkennung oder einer durch einen SIGN ON Aufruf übergebenen Benutzerkennung angemeldet. Ist der Benutzer unter der Verbindungs-Benutzerkennung angemeldet, so kann der Anmelde-Vorgang den Benutzer durch den Aufruf SIGN ON noch unter einer echten Benutzerkennung anmelden, wenn für die Verbindungs-Benutzerkennung kein Vorgang offen ist (KCRSIGN1=R).
- Bei UPIC-Clients wird der Benutzer unter der Verbindungs-Benutzerkennung oder der im UPIC-Protokoll übergebene Benutzerkennung oder der durch einen SIGN ON Aufruf übergebenen Benutzerkennung angemeldet. Ist der Benutzer unter der Verbindungs-Benutzerkennung angemeldet, so kann der Anmelde-Vorgang im Aufruf SIGN ON noch eine echte Benutzerkennung übergeben.

Der zweite Teil des Anmelde-Vorgangs

Mit dem Aufruf SIGN ST können Sie den Status abfragen.

1. KCRSIGN1= A oder R

   Die Anmeldung war erfolgreich. Der Vorgang ist jetzt einer Benutzerkennung zugeordnet.
   Nach dem Start eines Anmelde-Vorgangs einer Transportsystem-Anwendung ist die Verbindungs-Benutzerkennung angemeldet.
   Nach dem Start eines Anmelde-Vorgangs für einen UPIC-Client ist entweder die Verbindungs-Benutzerkennung oder eine vom Client im UPIC-Protokoll übergebene echte Benutzerkennung angemeldet. Ist der Client unter der Verbindungs-Benutzerkennung angemeldet, kann der Anmelde-Vorgang über den Aufruf SIGN ON jetzt eine echte Benutzerkennung übergeben.

   Bei KCRSIGN1 = A
   kann sich der Anmelde-Vorgang mit PEND FI oder PEND FC beenden. Die abschließende Nachricht erzeugt der Vorgang selbst und gibt sie mit MPUT aus. Wird der Anmelde-Vorgang mit MPUT PM, KCLM=0, KCMF=SPACE und PEND FI abgeschlossen, gibt openUTM die letzte Dialog-Nachricht aus und beendet die Conversation. Ist keine Nachricht vorhanden, wird eine "NULL-Nachricht" ausgegeben und die Conversation abnormal beendet.

   Bei KCRSIGN1 = R
   Die Anmeldung ist korrekt, es liegt ein Vorgangs-Wiederanlauf vor. Wenn gewünscht, können vor Vorgangsende weitere Dialog-Schritte eingeschoben werden. Der Anmelde-Vorgang muss sich mit PEND FI beenden. Der Vorgangs-Wiederanlauf wird durch den Aufruf MPUT PM, KCLM=0, KCMF/kcfn=Leerzeichen ausgelöst. In diesem Fall gibt openUTM die gesicherte letzte Nachricht des unterbrochenen Vorgangs aus (Bildschirmwiederanlauf) oder startet bei lokalem Sicherungspunkt nach PEND SP/ FC das Folgeteilprogramm bzw. den Folge-Vorgang.

   Durch Beendigung ohne MPUT-Aufruf kann der offene Vorgang abnormal beendet werden. Ein UPIC-Client erhält ein CM_DEALLOCATED_ABEND, an einen Transportsystem-Anwendung wird eine K017-Meldung gesendet.

2. KCRSIGN1= U

   Die Anmeldung ist nicht erfolgreich, d.h. openUTM hat die Berechtigungsdaten nicht akzeptiert. Beendet sich ein Anmelde-Vorgang für einen UPIC-Client in diesem Zustand, wird die Conversation beendet. Beendet sich der Anmelde-Vorgang für eine Transportsystem-Anwendung in diesem Zustand, so wird die Verbindung abgebaut.

Besonderheiten des Anmelde-Vorgangs für UPIC-Clients

Der Anmelde-Vorgang wird vor Beginn jeder Conversation gestartet, in Anwendungen mit OMIT-UPIC-SIGNOFF=NO aber nur, wenn der Benutzer noch nicht angemeldet ist.

Ein PEND FI im Anmelde-Vorgang nach erfolgreicher Anmeldung beendet den Anmelde-Vorgang, aber nicht die Conversation.

Wird ein Teilprogramm des Anmelde-Vorgangs nach Empfang einer Nachricht vom UPIC-Client mit PEND PA/PR, PS oder FC ohne vorherigen MPUT beendet, dann kann das im Feld KCRN spezifizierte Folgeteilprogramm noch nicht gelesene Nachrichten(-teile) lesen. Wird der Anmelde-Vorgang mit PEND FC ohne vorhergehenden MPUT beendet, so erhält das erste Teilprogramm des geketteten Vorgangs als Vorgangs-Kennzeichen im KBKOPF den Wert F (First), nicht C (Chained), da es eine Nachricht vom Client erhält.

Die folgende Abbildung zeigt ein Beispiel für den Ablauf eines erfolgreichen Anmelde-Vorgangs über einen UPIC-Client, der die Berechtigungsdaten einer echten Benutzerkennung im Protokollfeld übergibt.

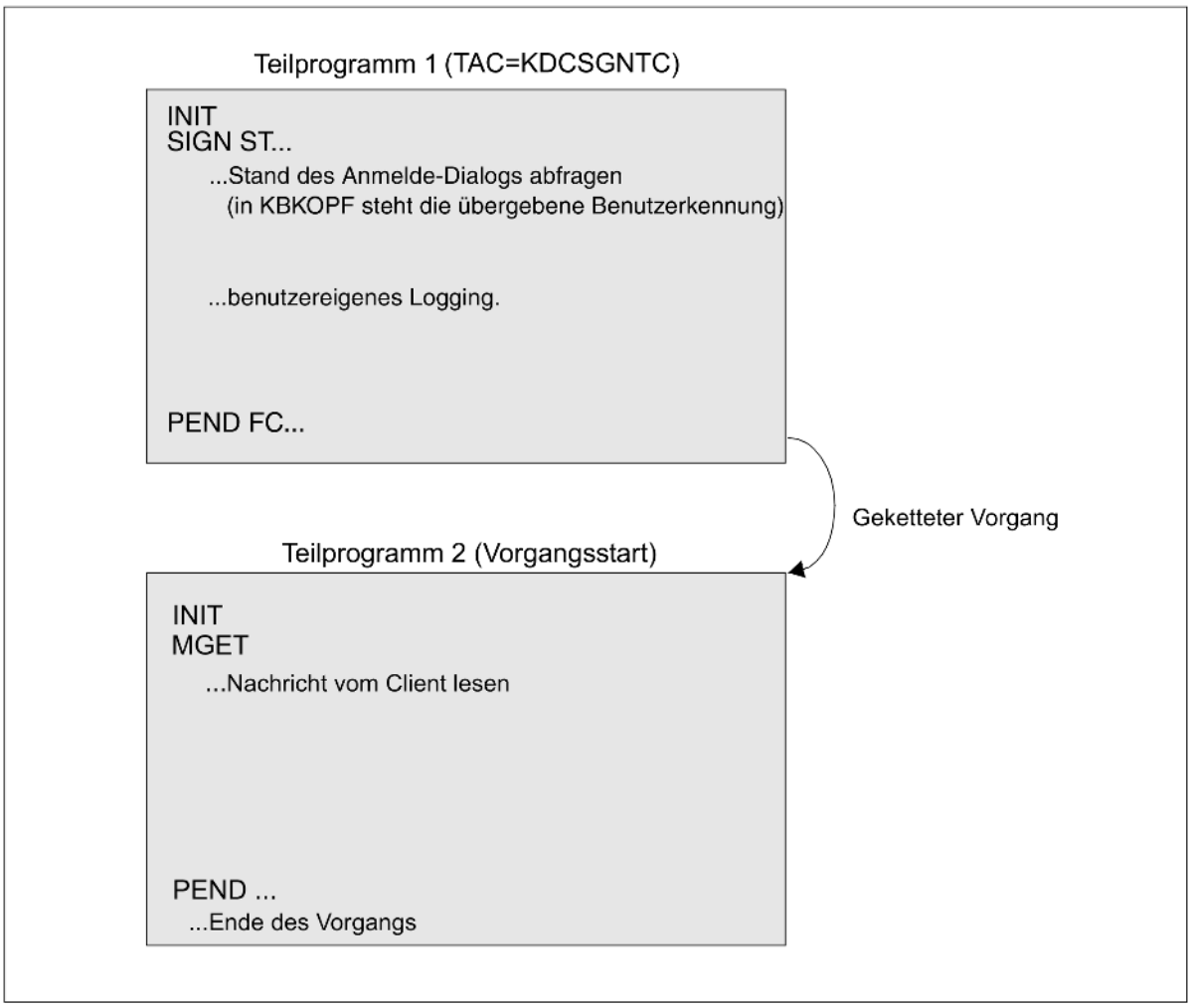


Bild: Ablauf eines Anmelde-Vorgangs für UPIC-Clients

In Teilprogramm 1 wird der Anmelde-Vorgang mit PEND FC beendet. Als Folge-TAC wird der beim SIGN ST übergebene Transaktionscode aus dem UPIC-Protokoll übernommen.

Der gekettete Vorgang kann dann in Teilprogramm 2 die Nachricht vom Client lesen. Auf diese Weise können UPIC-Clients den Anmelde-Vorgang nutzen, ohne dass sie umprogrammiert werden müssen.

Beispiel eines Anmelde-Vorgangs für TS-Anwendungen

Die folgende Abbildung zeigt den Ablauf eines Anmelde-Vorgangs über eine TS-Anwendung bei erfolgreicher Anmeldung unter einer echten Benutzerkennung:

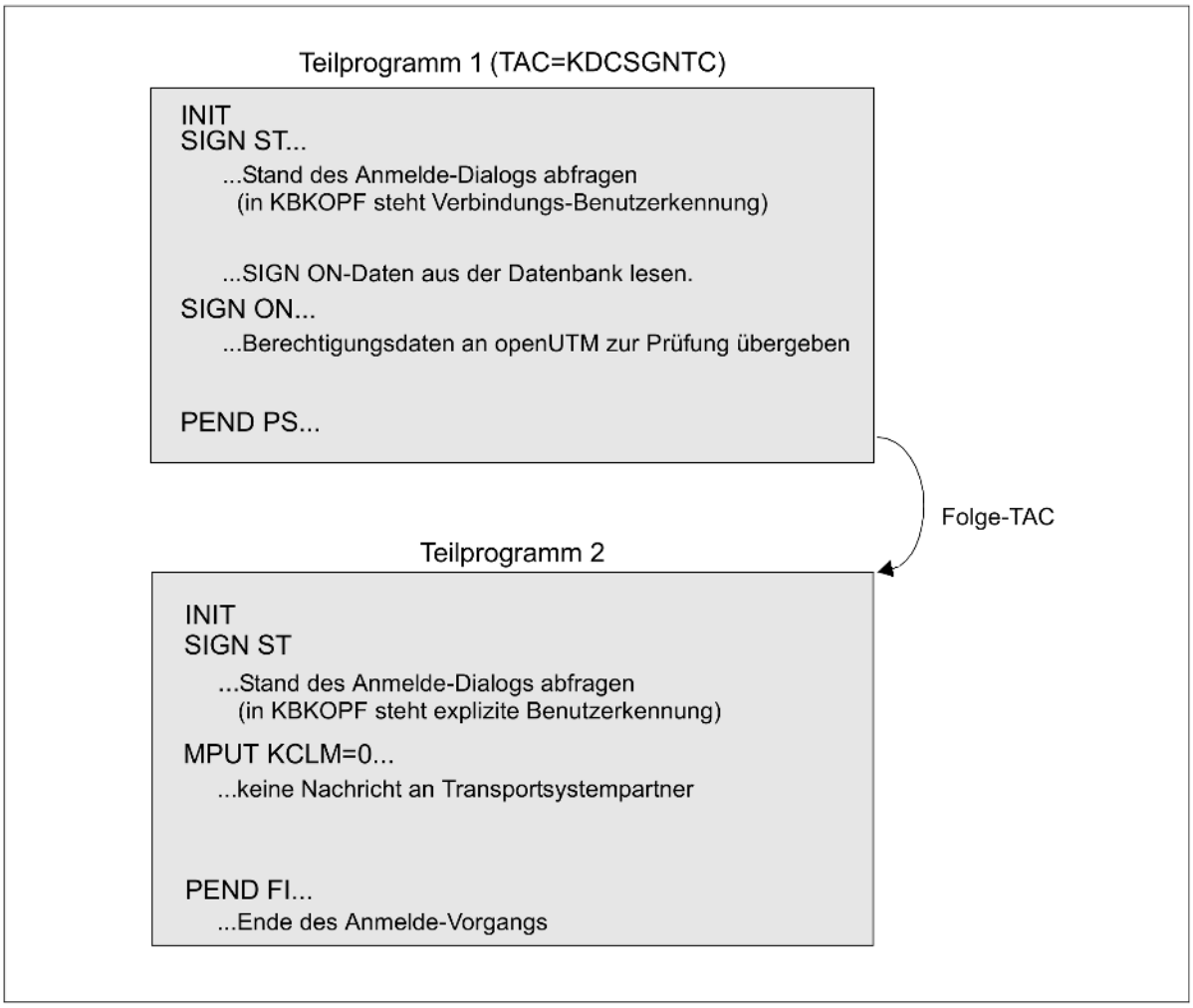


Bild: Ablauf eines Anmelde-Vorgangs für TS-Anwendungen

Beispielprogramme für Anmelde-Vorgang

Zusammen mit openUTM werden Teilprogramme als Quellprogramme ausgeliefert, die einen fertigen Anmelde-Vorgang mit formatierter Schnittstelle zum Terminal realisieren. Dieser Anmelde-Vorgang ist für alle Generierungsvarianten geeignet. Das verwendete Format enthält englische Texte.

Der Anwender kann diese Vorlage nach seinen Wünschen abändern und erhält so auf einfache Weise ein Anmeldeverfahren mit formatierter Schnittstelle zum Benutzer. Er muss dadurch mit der Programmierung nicht völlig neu beginnen.

10 Ergänzungen für C/C++

Dieses Kapitel ergänzt die allgemeinen Informationen der Kapitel „Aufbau und Einsatz von UTM-Programmen" bis „Event-Funktionen" um Sprachspezifische Informationen, die Sie speziell für die Erstellung von C- oder C++-Teilprogrammen benötigen.

Im ersten Abschnitt erfahren Sie, wie C/C++-Teilprogramme aufgebaut sind, der zweite Abschnitt enthält Programm-Beispiele und im dritten Abschnitt sind die Datenstrukturen *kcca.h*, *kcmac.h* und *kcpa.h* aufgelistet.

10.1 Programmaufbau bei C/C++-Teilprogrammen

In diesem Kapitel erfahren Sie

- wie ein UTM-Teilprogramm als Unterprogramm zu erstellen ist
- wie Sie die Daten deklarieren müssen
- wie der Befehlsteil aussehen muss und wie ein KDCS-Aufruf programmiert wird
- welche Plattform-spezifischen Besonderheiten bestehen (z.B. Abhängigkeiten von spezifischen Compilern, Formatierungssystemen)

10.1.1 C/C++-Teilprogramme als Unterprogramme

UTM-Teilprogramme und Event-Exits sind Unterprogramme der UTM-Main-Routine. Daraus ergeben sich folgende Konsequenzen:

- Der Programmname definiert die Einsprungadresse.
- Ein C/C++-Teilprogramm ist als Funktion vom Typ *void* zu definieren.
- Alle formalen Parameter müssen explizit deklariert werden.
- Das Teilprogramm wird dynamisch mit dem PEND-Aufruf beendet; eine Ausnahme bilden die Event-Exits, die mit der Anweisung *return* verlassen werden. Die Anweisung *exit* ist grundsätzlich verboten.

Um kompatibel zu sein und mit korrekten Datenstrukturen zu arbeiten, stehen Ihnen eine Reihe von Include-Dateien (Header-Files) zur Verfügung. Für C und für C++ verwenden Sie dieselben Include-Dateien. Die Verwendung dieser Include-Dateien wird in Abschnitt „Datenstrukturen für C/C++-Teilprogramme“ beschrieben. Die Include-Dateien *kcca.h*, *kcmac.h* und *kcpa.h* finden Sie:

- auf BS2000-Systemen in der Bibliothek *SYSLIB.UTM.070.C*,
- auf Unix-, Linux- und Windows-Systemen im Verzeichnis *include* im UTM-Verzeichnis *utmpfad*.

> Falls Sie C-Programme nicht übersetzen können, weil der erste Aufrufparameter von KDCS statt dem Typ *union kc_paa ** den Typ *struct kc_pa ** hat, setzen Sie für den C-Precompiler die Option UTM_OLDANSI, damit die Prüfung unterbleibt.

Teilprogrammname

Der Name eines C/C++-Teilprogramms ist auch seine Einsprungadresse.

Dieser Name ist frei wählbar. Er muss innerhalb eines Anwendungsprogramms eindeutig sein. Einige Namen sind bereits vergeben und dürfen daher nicht verwendet werden.

Bei der Namenswahl sollten Sie deshalb folgende Punkte beachten:

- Für BS2000-Systeme:
  - Alle Namen, die mit KDC, KC oder I beginnen, sind reserviert und sollten vermieden werden.
- Für Unix-, Linux- und Windows-Systeme:
  - Alle Namen, die mit KDC, KC , x oder ITS beginnen, sind reserviert.
  - Namen, die mit t_ beginnen, sind für PCMX reserviert.
  - Namen, die mit a_, o_ oder s_ beginnen, sind für OSS reserviert.
- Der Name muss den C/C++-Konventionen entsprechen.

Den Programmnamen (Einsprungnamen) müssen Sie auch bei der Generierung der UTM-Anwendung angeben, und zwar jeweils in der KDCDEF-Anwendung PROGRAM
(siehe openUTM-Handbuch „Anwendungen generieren“).

10.1.2 Parameter eines C/C++-Teilprogramms

Ein C/C++-Teilprogramm besitzt mindestens einen, in der Regel jedoch mehrere Parameter, die wie folgt in Form von Adressen übergeben werden.

```
[extern "C"] void cprog (kb [,spab] [,param_1] ....[,param_n])
```

extern "C"	nur notwendig bei C++-Teilprogrammen: C++-Teilprogramme müssen sich in ihren Source-Codes gegenüber openUTM als extern "C" linkage identifizieren, sonst gibt es Bindefehler.
void	Ein C/C++-Teilprogramm ist als Funktion vom Typ *void* zu definieren.
cprog	Name des Teilprogramms. Er muss bei der Generierung in der PROGRAM-Anweisung angegeben werden, siehe openUTM-Handbuch „Anwendungen generieren“.
kb	Name des Kommunikationsbereichs (KB). Diesen Namen können Sie frei wählen, aber der KB muss anschließend unter diesem Namen deklariert werden. Als Include-Element steht Ihnen *kcca.h* zur Verfügung.
spab	Name des Standard-Primären-Arbeitsbereichs. Diesen Namen können Sie frei wählen, aber der SPAB muss anschließend unter diesem Namen deklariert werden. Als Include-Element für den KDCS-Parameterbereich steht Ihnen *kcpa.h* zur Verfügung.
param_1 ... param_n	sind die Namen weiterer Objekte (AREAs), die ebenfalls deklariert werden müssen. Diese Objekte können insbesondere Speicherbereiche sein, die als Erweiterung des SPAB dienen. Werden diese Objekte nicht verwendet, so entfällt die Angabe.

Bei der Verwendung von *kcmac.h* werden die Include-Dateien *kcca.h*, *kcpa.h*, *kcapro.h* und *kcdf.h* implizit inkludiert. D.h. sie müssen nicht mehr im eigentlichen Programm als Include-Dateien angegeben werden. Ihre Definitionen stehen dem Programmierer jederzeit zur Verfügung.

Für die Übergabe von Leerzeichen an Char-Arrays ist in der Include-Datei `kcmac.h` der Wert KDCS_SPACES definiert, für die Übergabe des Wertes "binär Null" an Char-Parameter der Wert KDCS_NULL.

10.1.3 Datendeklaration

Sie müssen alle formalen Parameter explizit deklarieren. Beachten Sie dabei nachfolgend beschriebene Punkte.

10.1.3.1 Kommunikationsbereich

Jedes Teilprogramm einschließlich der Event-Exits (Ausnahme: INPUT-Exit) muss eine Datenstruktur enthalten, die den KDCS-Kommunikationsbereich (KB) beschreibt. Verwenden Sie dazu die Include-Datei *kcca.h*.

10.1.3.2 Standard Primärer Arbeitsbereich (SPAB)

In der Regel enthält ein C/C++-Teilprogramm auch eine Datenstruktur für den Standard-Primären-Arbeitsbereich (SPAB). Wird der SPAB verwendet, sollte er den KDCS-Parameterbereich (Include-Datei *kcpa.h*) enthalten. Auch die Nachrichtenbereiche und andere variable Daten sollten Sie in den SPAB legen.

Wenn Sie variable Daten nicht in den SPAB legen, dann müssen Sie dafür sorgen, dass das Teilprogramm reentrant-fähig ist, z.B. indem Sie sie im automatic-Bereich der Funktion ablegen.

Die Nachrichtenbereiche müssen Sie selbst definieren. Für Aufrufe, die Informationen von openUTM anfordern (z. B. KDCS_INFOSI, KDCS_INITPU) stehen jedoch spezifische Datenstrukturen in Include-Dateien zur Verfügung. Falls Sie mit einem
Formatierungssystem arbeiten, können Sie für die Strukturierung des Nachrichtenbereichs automatisch generierte Adressierungshilfen verwenden (siehe Handbuch des Formatierungssystems).

Im folgenden Beispiel enthält der Kommunikationsbereich auch einen KB-Programmbereich zur Datenübergabe an Folgeteilprogramme. Der Nachrichtenbereich wird in den SPAB gelegt.

Beispiel

```
#include <kcmac.h>                        /* UTM-Datenstrukturen        */
#include <forma3a.h>                      /* Struktur der Adressierungs-*/
                                          /* hilfe für +Format forma3   */
  struct ca_area
   { struct ca_hdr ca_head;               /* KB-Kopf                    */
     struct ca_rti ca_return;             /* KB-Rueckgabebereich        */
     struct ca_prog_area
      { char ca_info[22];                 /* Anwenderspezifische        */
        char ca_start[2];                 /* Deklaration des KB-        */
        char ca_dest[2];                  /* Programmbereichs           */
        char ca_fl_day[5];
        char ca_fl_nr1[5];
        char ca_fl_nr2[5];
      } ca_prg;
   };
  struct work
   { union kc_paa param;                  /* KDCS-Parameterbereich      */
     struct msg_area
      { forma3a std_mask;                 /* Deklaration +Format forma3 */
        ...
      } msg_a;                            /* Nachrichtenbereich         */
   };
void cprog (struct ca_area *ca, struct work *spab)
{.../* Beginn Funktionsbereich des Teilprogramms */ ...
```

10.1.3.3 Weitere Datenbereiche (AREAs)

Zusätzlich zum Kommunikationsbereich und zum SPAB können Sie noch weitere Bereiche als Parameter übergeben, siehe "Weitere Bereiche". Einen solchen Bereich legen Sie in C/C++ als Quelltext an, welcher nur Datendefinitionen, aber keine ausführbaren Anweisungen enthält.

Wie Sie solche Bereiche in C/C++ definieren und in Ihren C/C++-Programmen einsetzen, wird mit Hilfe von Beispielen erläutert.

Beispiel mit AREAs (Unix-, Linux- und Windows-Systeme)

Im Folgenden werden zwei Areas generiert, in einer C-Source definiert und an ein Teilprogramm übergeben. Definiert werden:

- die Area *area* für den direkten Zugriff, d.h. der Datenbereich wird direkt an das Teilprogramm übergeben, und
- die Area *areaind* für den indirekten Zugriff.

KDCDEF-Generierung

```
AREA area,ACCESS=DIRECT
AREA areaind,ACCESS=INDIRECT
```

C/C++-Source *zur Versorgung der zusätzlichen Datenbereiche erstellen*

Die zu den Areas gehörenden Datenstrukturen definieren Sie wie folgt in einer C/C++-Source. In diesem Beispiel wird bei *areaind*, d.h. bei der Area mit indirektem Zugriff, die Adresse der Area zum Übersetzungszeitpunkt gesetzt.

Definition von Areas

```
char area[20] = "Area direct ";
static char area_ind[30] = "Area indirect ";
char *areaind = &area_ind[0];
```

Die C/C++-Source müssen Sie mit dem C/C++-Compiler übersetzen und das erzeugte Objektmodul müssen Sie zu den Teilprogrammen dazu binden.

Wollen Sie die Adresse der Area *areaind* während des Anwendungslaufs setzen, dann definieren Sie *areaind* in der C/C++-Source wie folgt:

```
char *areaind;
```

Während des Anwendungslaufs (typischerweise im Event-Exit START) müssen Sie dann *areaind* mit der Adresse der Area versorgen, den Sie den Teilprogrammen als Parameter übergeben wollen, z.B. durch die Anweisung:

```
static char area_ind[30] = "Area indirect ";
areaind = &area_ind[0];
```

Damit ist es beispielsweise möglich, im Event-Exit START ein Shared Memory zu eröffnen und die Adresse des Shared Memory in der Zeigervariablen *areaind* zu hinterlegen. Die Teilprogramme bekommen damit die Möglichkeit, auf das Shared Memory zuzugreifen.

Zugriff auf die Area-Bereiche in den Teilprogrammen auf Unix-, Linux- und Windows-Systemen

Ein Teilprogramm, in dem Sie die Areas bzw. einer der Areas benutzen, muss wie folgt aussehen. Dabei ist zu beachten, dass die Reihenfolge der Areas in der Parameterliste mit der Reihenfolge der AREA-Anweisungen bei der KDCDEF-Generierung übereinstimmen muss.

Teilprogramm in C

```
void areaprg (
struct spab *spab ,
struct kc_ca *kb ,
char area1 [20] ,
char area2 [30] )
{
sprintf ( BUFFER
, "Hello world from UTM (Lterm = %.8s)\n"
"Area is ’%s’ \n"
"Area (indirect) is ’%s’ \n"
, kb -> kopf.kclogter
, area1
, area2
);
...
```

Alternative zu AREAs

Falls Teilprogramme, die AREAs verwenden, aus einer Anwendung in eine andere Anwendung übernommen werden sollen, kann die Verwendung von AREAs auf Grund ggf. unterschiedlicher Parameterleisten zu Problemen führen.

Für C/C++-Programme gibt es folgende Alternativen. Man unterscheidet hier, ob der Datenbereich im prozesslokalen Speicher oder im Shared Memory auf dem Unix- oder Linux-System bzw. im Memory Mapped File auf dem Windows-System liegt.

BS2000-Systeme

Für BS2000-Systeme: Datenbereiche im prozesslokalen Speicher
Hierzu muss man nur für jeden Datenbereich in C/C++ eine Struktur global zur Verfügung stellen, auf die andere Module mit der gleichen Definition und zusätzlich dem Speicherklassenattribut 'extern' zugreifen können.

Unix-, Linux- und Windows-Systeme

Für Unix- und Linux-Systeme im Shared Memory und für Windows-Systeme im Memory Mapped File:

Im Startexit stellt man eine Struktur zur Verfügung (siehe Beispiel 3) mit:

```
static DataAreas struct {
       struct table *TABLE1;
       struct table *TABLE2;
       struct table *TABLE3;
       };
...
struct DataAreas ExternDataAreas;
```

Im Startexit werden dann die Shared Memory-Bereiche angefordert und die Adressen in der Struktur ExternDataAreas versorgt.
Zugegriffen wird von anderen Teilprogrammen auf die Struktur

```
extern struct DataAreas ExternDataAreas;
```

Beispiel mit AREAs (BS2000-Systeme)

Man stellt einen Datenbereich (siehe Beispiel), der z.B. in Assembler geschrieben worden ist, zur Verfügung:

```
TABLE1   CSECT PUBLIC
         DS    CL64
         END
```

Diese CSECT wird übersetzt und ggf. mit weiteren Modulen gebunden. Das Modul (ohne ein weiteres dazugebundene Modul mit Coding) soll den Namen MTABLE1 haben, dann generiert man es als Lademodul:

```
LOAD-MODULE  MTABLE1
           , LOAD-MODE = (POOL , poolname , NO-PRIVATE-SLICE ) -
           , ...
```

In einem Teilprogramm oder Event-Exit ermöglichen Sie dann mit

```
extern struct table TABLE1;
```

den Zugriff auf den Datenbereich. Dieses Teilprogramm oder Event-Exit muss dann dynamisch nachgebunden werden.

Beispiel mit AREAs (alle Systeme)

Die Bereiche TABLE1, TABLE2 und TABLE3 sind in dieser Reihenfolge mit der AREA-Anweisung definiert worden. In einem Teilprogramm wird TABLE3 benötigt. TABLE1, TABLE2 und TABLE3 haben die gleiche Struktur und werden wie folgt definiert:

```
struct TABLE
   { int   nr;
     int  tag;
     char name[20];
     char firma[20];
     int best_nr;
     int menge;
     float preis;
     int rabatt;

};
```

Die Adressen dieser Bereiche werden wie folgt übergeben.

```
 ...
#include <kcmac.h>                        /* UTM-Datenstrukturen    */
   ...
   struct ca_area
     {...};
   struct work
     {...};
struct TABLE
   {...};
...
void cprog (struct ca_area *ca, struct work *spab, struct TABLE *TABLE1,
            struct TABLE *TABLE2, struct TABLE *TABLE3)
{ ...
```

10.1.4 Datenstrukturen für C/C++-Teilprogramme

Um die Datenbereiche zu strukturieren, werden mit openUTM folgende Include-Dateien (Header-Files) ausgeliefert, die vordefinierte Datenstrukturen enthalten:

Auf BS2000-Systemen sind die Datenstrukturen in der Bibliothek SYSLIB.UTM.070.C enthalten.

Auf Unix-, Linux und Windows-Systemen sind die Datenstrukturen im Verzeichnis *include* im UTM-Verzeichnis *utmpfad* enthalten.

Name	Inhalt und Bedeutung
kcapro.h	optionaler zweiter Parameterbereich für den APRO-Aufruf: Dieser Bereich dient zur Auswahl spezieller OSI TP-Funktionskombinationen und des Security-Typs. *kcapro.h* wird durch *kcmac.h* abgesetzt.
kcat.h	KDCS-Attributfunktionen (nur auf BS2000-Systemen): Bei Verwendung von +Formaten können Sie mit den symbolischen Namen für Attributfunktionen die Attributfelder Formate verändern.
kcca.h	Datenstruktur für den KDCS-Kommunikationsbereich (communication area); dieser enthält: • aktuelle Daten des Vorgangs und Programms, • Rückgaben nach einem Aufruf an openUTM und • falls gewünscht den KB-Programmbereich zur Datenübergabe zwischen Programmen in einem Vorgang. Die Felder des KB-Programmbereichs müssen Sie zusätzlich definieren. *kcca.h* wird durch *kcmac.h* abgesetzt.
kccf.h	Nur auf BS2000-Systemen: definiert den zweiten Parameter, den openUTM beim Event-Exit INPUT übergibt. In diesem Parameter übergibt openUTM die Inhalte der Steuerfelder von Bildschirmformaten an das Teilprogramm. Dieser zweite Parameter wird deshalb auch Steuerfeldbereich (Control Fields) genannt.
kcdad.h	Datenstruktur für den DADM-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf DADM RQ über den Nachrichtenbereich legen.
kcdf.h	KDCS-Bildschirmfunktionen: Mit diesen symbolischen Namen können Sie die Bildschirmausgabe beeinflussen, indem Sie den Namen der gewünschten Funktion ins Feld KCDF des KDCS-Parameterbereiches bringen. *kcdf.h* wird durch *kcmac.h* abgesetzt.
kcinf.h	Datenstruktur für den INFO-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf INFO DT/SI/PC über den Nachrichtenbereich legen.
kcini.h	definiert einen zweiten Parameterbereich für den INIT-Aufruf (nur notwendig bei INIT PU): In diesen Parameterbereich liefert openUTM die mit dem INIT PU-Aufruf angeforderten Informationen zurück.
kcinp.h	Datenstruktur für den INPUT-Exit: Diese Datenstruktur enthält die Eingabe- und Ausgabeparameter des INPUT-Exits.

Name	Inhalt und Bedeutung
kcmac.h	KDCS-Makro-Schnittstelle für C/C++: Diese Datei enthält alle Makros der C/C++-Makro-Schnittstelle sowie die Include-Anweisungen für die Include-Dateien *kcapro.h*, *kcca.h, kcdf.h* und *kcpa.h*.
kcmsg.h	Datenstruktur für die UTM-Meldungen: Diese Datenstruktur benötigen Sie, wenn Sie Meldungen in einer MSGTAC-Routine behandeln oder wenn Sie die SYSLOG-Datei mit einem eigenen Programm auswerten wollen.
kcpa.h	Datenstruktur für den KDCS-Parameterbereich: Dieser Bereich nimmt die Parameter eines KDCS-Aufrufs auf. *kcpa.h* wird durch *kcmac.h* abgesetzt.
kcpad.h	Datenstruktur für den PADM-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf PADM AI/PI über den Nachrichtenbereich legen.
kcsgst.h	Datenstruktur für den SIGN-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf SIGN ST mit KCLA > 0 über den Nachrichtenbereich legen.

Diejenigen Datenstrukturen, die Sie verwenden, fügen Sie vor dem Aufruf des Teilprogramms per *#include* ein. Im Teilprogramm müssen Sie die entsprechenden Bereiche (Kommunikationsbereich, KDCS-Parameterbereich,...) explizit deklarieren.

Beispiel 3

```
/* Konstanten und Datenstrukturen einfügen */
#include <kcmac.h>                        /* UTM-Datenstrukturen        */
#include <kcinf.h>
   struct ca_area {...};
   struct work
    { union kc_paa param;
      struct msg_area
       { struct kc_dttm   info_time;      /* Bereich fuer INFO DT       */
         struct kc_sysinf info_sys;       /* Bereich fuer INFO SI       */
         char text[200];
       } msg_a;
    };
void cprog (struct ca_area *ca, struct work *spab)
```

10.1.5 Befehlsteil eines C/C++-Teilprogramms

Den Befehlsteil eines C/C++-Teilprogramms können Sie frei gestalten. Nur wenige Regeln der Transaktionsverarbeitung, wie sie in Kapitel „Aufbau und Einsatz von UTM-Programmen“ ausführlich beschrieben sind, müssen Sie beachten:

- Die Teilprogramme sind Unterprogramme der UTM-Main Routine KDCROOT.
- Die Teilprogramme müssen reentrant-fähig sein.
- Dialog-Teilprogramme müssen den strengen Dialog einhalten.

Mit KDCROOT bezeichnet man die UTM-Main Routine. Das Quellprogramm für KDCROOT wird mit dem Generierungstool KDCDEF erzeugt, siehe openUTM-Handbuch „Anwendungen generieren“.

Für Event-Exits gelten besondere Regeln, die in Abschnitt „Event-Exits“ beschrieben werden.

Lokale Klassen in C++-Teilprogrammen

Wird in einem C++-Teilprogramm eine lokale Klasse deklariert, kann der Destruktor zu dieser Klasse nur ablaufen, wenn sich die Klasse in einem Block befindet, dessen Blockende "}" vor dem PEND-Aufruf ist. Diese Einschränkung gilt nicht unter Linux-Systemen.

Empfehlung: Für lokale Klassen sollten Sie einen eigenen "inneren" Block verwenden.

Beispiel

```
//extract of cpphello.C in sample Application
extern "C" void cpphello (struct kc_ca *kb, struct work *spab)
{
  { Demo Autoclass('A');
     // further code using Autoclass
  }
  // reached after destructure call for Autoclass
:
:
  /* PEND-FI - Call */
  KDCS_PENDFI();
}
```

KDCS-Aufrufe in C/C++Teilprogrammen

Für den Aufruf von UTM-Funktionen bietet Ihnen KDCS die Nutzung der C/C++-Makroschnittstelle, die eine komfortable Versorgung der Parameter ermöglicht (siehe "C/C++-Makroschnittstelle").

10.1.6 C/C++-Makroschnittstelle

Um die Parameterübergabe an die KDCS-Schnittstelle zu vereinfachen, werden in der Include-Datei *kcmac.h* Makros für die einzelnen KDCS-Aufrufe zur Verfügung gestellt. Diese Makros enthalten alle für einen KDCS-Aufruf benötigten Angaben als Makroparameter. Sie führen den gewünschten Aufruf aus und stellen danach einen Returncode zur Verfügung (siehe auch "Programmierung der KDCS-Fehlerbehandlung").

In den folgenden Abschnitten werden die Namengebung für die Makros, sowie die Namen und die Eigenschaften der verschiedenen Makroparameter erklärt. Es wird erläutert, welche vorbereitenden Maßnahmen getroffen werden müssen, um die Include-Datei *kcmac.h* einsetzen zu können. Am Ende des Abschnitts finden Sie ein Beispiel für einen Makro-Aufruf und das Listing eines ablauffähigen KDCS-Programms, das den Einsatz der Include-Datei *kcmac.h* verdeutlicht.

KDCS_SET zur Vorbereitung

In jeder Übersetzungseinheit, in der Sie einen KDCS-Makro aufrufen wollen, müssen Sie vor dem ersten Makro-Aufruf die folgenden beiden Aktionen ausführen:

- die Include-Datei *kcmac.h* in die Übersetzungseinheit kopieren

  Bei der Verwendung von *kcmac.h* werden die vier Include-Dateien *kcca.h*, *kcpa.h*, *kcapro.h* und *kcdf.h* implizit verwendet, d.h. diese Definitionen stehen dem Programmierer jederzeit zur Verfügung.
- das Makro KDCS_SET(pb,hdr,rti) zur Initialisierung aufrufen

  pb: Zeiger auf den KDCS-Parameterbereich.

  hdr: Zeiger auf den Kopfbereich des Kommunikationsbereichs (Struktur ca_hdr).

  rti: Zeiger auf den Rückgabebereich des Kommunikationsbereichs (Struktur ca_rti).

Der KDCS_SET-Aufruf stellt die Verbindung zwischen den KDCS-Makros und den Definitionen des Programmierers für Parameterbereich und Kommunikationsbereich her. Sie können dieses Makro auch mehrfach aufrufen. So ist es beispielsweise möglich mehrere verschiedene Parameterbereiche zu verwenden.

Makronamen

Die Namen der Makros ergeben sich nach folgenden Regeln:

Ein KDCS-Makroname beginnt immer mit dem Präfix "KDCS_". Es folgt der Operationscode des gewünschten KDCS-Aufrufs in Großbuchstaben. An diesen wird, falls benötigt, die Operationsmodifikation angehängt (ebenfalls in Großbuchstaben).

Beispiele:

KDCS_INIT, KDCS_LPUT, KDCS_MGET

KDCS_MPUTNT, KDCS_PENDFI, KDCS_SPUTMS

Eine Ausnahme bilden dabei die DPUT-Aufrufe zusammen mit einem "+T", "-T", "+I" oder "-I" als kcom-Parameter. Diese Makros heißen: KDCS_DPUTPT, KDCS_DPUTMT, KDCS_DPUTPI bzw. KDCS_DPUTMI.

Eine weitere Ausnahme bilden die APRO-Aufrufe zur Adressierung von OSI TP-Partnern: KDCS_APRODM_OSI, KDCS_APROAM_OSI, KDCS_APRODM_OSI_O, KDCS_APROAM_OSI_O

Makroparameter

Die Makroparameter sind jeweils nach den KDCS-Parametern benannt, die sie mit Werten versorgen sollen (*kcrn*, *kclt*, *kchour*, *kcpi*,...). Der Nachrichtenbereich heißt *nb* und wird, falls benötigt, immer als erster Parameter angegeben.

Man unterscheidet zwischen drei verschiedenen KDCS-Parametertypen: Char-Arrays, Zeichen und Zahlen:

- Char-Arrays

  An der KDCS-Schnittstelle haben diese Parameter die Länge zwei, drei oder acht Zeichen. An der C/C++-Makro-Schnittstelle werden diese Parameter als Zeiger auf einen C-String beliebiger Länge angegeben.
  Intern werden diese Parameter in Char-Arrays der benötigten Länge umgewandelt. Entweder durch Anhängen eventuell fehlender Leerzeichen, oder durch Ignorieren überflüssiger Zeichen an Ende des C-Strings. Wird daher ein Array der Länge acht z.B. mit drei Leerzeichen "   " oder keinem Zeichen "" versorgt, dass mit Leerzeichen auf acht Leerzeichen "   " aufgefüllt wird. Bei der Angabe "xyz" wird mit fünf Leerzeichen zu "xyz   " aufgefüllt. Die Umwandlung erfolgt jeweils ohne dass ein Stringendekennzeichen "\0" erzeugt wird.

  Parameter dieses Typs: *kcrn, kcfn, kclt, kcpa, kcus, kcadrlt, kcact, kcpi, kcpos, kcneg, kccomid, kclangid, kcterrid, kccsname*.

- Zeichen

  An der KDCS-Schnittstelle sind diese Parameter vom Typ Character.
  An der C/C++-Makro-Schnittstelle werden diese Parameter auch als character (per value) angegeben. Die Angaben sind daher von der Form: ' ', 'A', 'C', usw.

  Parameter dieses Typs: *kcmod*, *kcof.*

- Zahlen

  An der KDCS-Schnittstelle gibt es verschiedene Arten von Zahlen: short, unsigned short und Zahlen, die in Form eines abdruckbaren Texts erwartet werden. An der C/C++-Makro-Schnittstelle werden diese Parameter immer als Zahlen (per value) angegeben. Diese Zahlen werden intern auf das gewünschte Format umgewandelt. Short und unsigned short Zahlen werden direkt übergeben. Zahlen, die als Text bereitgestellt werden müssen, werden umgewandelt.

  Parameter dieses Typs: *kcla, kclm, kcdf, kclcapa, kclspa, kcday, kchour, kcmin, kcsec, kcli*.

Vereinfachte Parameterübergabe

Die Makros kennen die geforderte Länge jedes Parameters und sorgen für eine korrekte Übergabe (Kürzere Strings werden bis zur erforderlichen Länge mit Leerzeichen aufgefüllt). Ein Parameter wird nur noch in der Makroparameterliste angegeben und muss nicht mehr mit *memcpy* übertragen werden. Die Makros verarbeiten auch alle Angaben, die bisher bei *memcpy* verwendet wurden. Die Zuweisung der Parameter auf die entprechenden KDCS-Parameterfelder erfolgt implizit durch die C/C++-Makros. Die Parametertypen werden durch die C/C++-Makro-Definitionen festgelegt.

Zum Vergleich ein Beispiel für die Versorgung des Parameters *kcrn*:

- Bei unmittelbarem Aufruf:

  ```
  memcpy(pb.kcrn,"rnam",8)  /* nur Versorgung von kcrn*/
  ```

- Bei Verwendung eines Makros:

  ```
  Makroname(...,"rnam",...) /* vollstaendiger KDCS-Aufruf*/
  ```

Nicht verwendete Parameterfelder werden implizit mit binär null versorgt. Beim Aufruf geben Sie für Felder, die mit Leerzeichen versorgt werden sollen, die Konstante KDCS_SPACES an.

Einige der KDCS-Parameter, die bisher auch als Char-Array übergeben werden mussten, liegen in C üblicherweise als Integer-Werte vor. Dies sind zum Beispiel die Parameter für die Zeitangabe: *kcday*, *kchour*, *kcmin*, *kcsec*. Solche Parameter werden in den neuen Makroaufrufen als Zahlen (integer) erwartet. Die Makros sorgen für eine korrekte Übergabe an die Schnittstelle.

Zum Vergleich die Versorgung eines KDCS-Aufrufs mit einer Zeitangabe:

- Bei unmittelbarem Aufruf:

```
memcpy(pb.kcext.kcdput.kcday,"003",3);
memcpy(pb.kcext.kcdput.kchour,"11",2);
memcpy(pb.kcext.kcdput.kcmin,"55",2);
memcpy(pb.kcext.kcdput.kcsec,"00",2);
```

- Bei Verwendung eines Makros:

```
Makroname(...,3,11,55,0)
```

Format des KDCS-Aufrufs über die C/C++-Makroschnittstelle

KDCS-Aufrufe über die C/C++-Makroschnittstelle haben folgendes Format:

KDCS_ *operationscode* [*operationsmodifikation*] (*parameterliste*)

Falls ein Nachrichtenbereich benötigt wird, ist dessen Adresse immer als erster Parameter zu übergeben. Die Parameterliste kann auch leer sein.

Bei der Bildung der Makronamen gibt es einige Ausnahmen (siehe "C/C++-Makroschnittstelle").

Beispiel

```
KDCS_INIT(länge_KB_programmbereich, länge_SPAB);
KDCS_SGETRL(zeiger_auf_NB, nachrichtenlänge, LSSB_Name)
KDCS_PGWTKP()
```

Beispiel: Makroaufruf KDCS_MPUTNT

Nachfolgend wird die Verwendung der KDCS-Makros am Beispiel eines MPUT NT- Aufrufs gezeigt. Zunächst die Beschreibung des Makros:

KDCS_MPUT NT(nb,kclm,kcrn,kcfn,kcdf)

char *nb	Zeiger auf Nachrichtenbereich
short kclm	Länge
char kcrn[8]	Leerzeichen/TAC/Vorgangs-ID
char kcfn[8]	Format/Leerzeichen/Editprofil
unsigned short kcdf	Bildschirmfüllzeichen / bin.null / ---

Das Makro KDCS_MPUTNT verlangt fünf Parameter, unter anderem einen Zeiger auf den Nachrichtenbereich *nb*. Dieser wird immer als erster Parameter übergeben. Die weiteren Parameter des Makros stammen aus dem KDCS-Parameterbereich. Die erforderlichen Typen und Bedeutungen der Parameter werden jeweils angegeben (z.B. ist *kclm* vom Typ *short* und gibt die Länge an, in der die Nachricht übermittelt werden soll). Im folgenden Beispiel ist NB ein Zeiger auf den verwendeten Nachrichtenbereich und *kc_pa* der KDCS-Parameterbereich.

Ein typischer Makroaufruf sieht z.B. wie folgt aus:

```
KDCS_MPUTNT(NB,10,KDCS_SPACES,KDCS_SPACES,KCNODF);
```

Derselbe Aufruf ohne Makro (unmittelbarer Aufruf der KDCS-Schnittstelle):

```
memcpy(PB.kcop,MPUT,4);
memcpy(PB.kcom,NT,2);
kc_pa.kclm=10;
memcpy(kc_pa.kcrn,"        ",8);
memcpy(kc_pa.kcfn,"        ",8);
kc_pa.kcdf=0;
KDCS(&kc_pa,NB);
```

DEBUG-Funktion

Sie können für die KDCS-Aufrufe, die durch die Makros abgesetzt werden, zur Laufzeit eine Protokollierung einschalten. Es werden das aufrufende Modul, die Quellzeile, und die KDCS-Felder KCOP, KCOM, KCRCCC und KCDCDC protokolliert.

Die Protokollierung wird wie folgt eingeschaltet:

- Unix-, Linux- und Windows-Systeme

  Durch Setzen und Exportieren der Umgebungsvariablen KDCS_C_DEBUG. Die Protokollierung erfolgt auf *stdout.*

- BS2000-Systeme

  Durch Setzen des Jobvariablen-Links *KDCSCDB.
  Die Protokollierung erfolgt auf SYSOUT.
  Bitte beachten Sie, dass der Jobvariablen-Link auf eine Jobvariable verweisen muss, die einen nicht-leeren Inhalt hat.

  Beispiel

  ```
  /CREATE-JV   JV-NAME = KDCSCDB
  /MODIFY-JV   JV-CONTENTS = KDCSCDB, SET-VALUE = 'YES'
  /SET-JV-LINK LINK-NAME = *KDCSCDB, JV-NAME = KDCSCDB
  ```

Weitere Informationen hierzu finden Sie im openUTM-Handbuch „Meldungen, Test und Diagnose“.

Hinweis: Makros als Statement-Folgen

Es handelt sich bei den KDCS-Makros nicht um einzelne C/C++-Funktionen. Vielmehr handelt es sich um eine Folge von mehreren Statements. Das bedeutet, dass ein einzelnes Makro genau so behandelt werden muss, wie es eine Folge mehrerer Statements erfordert. Normalerweise ist diese Eigenart nicht relevant, und die Makros können wie normale C/C++-Funktionen benutzt werden. Es gibt allerdings Programmstrukturen, die einzelne Statements erfordern, z.B. innerhalb eines if-Statements:

```
if (Bedingung) STATEMENT else STATEMENT;
```

Da die Makros aus mehreren einzelnen Statements bestehen, ist es nicht ohne weiteres möglich, sie für ein (einzelnes) STATEMENT einzusetzen. Sie müssen ein Makro daher zuerst als einen zusammengehörigen Block kennzeichnen. Dies geschieht mittels geschweifter Klammern: {Makro;}. Statt also fälschlicherweise zu schreiben

```
if (Bedingung) Makro; else Makro;      /* F A L S C H !!!! */
```

ist es richtig Folgendes zu verwenden:

```
if (Bedingung) {Makro;} else {Makro;}  /* R I C H T I G !! */
```

10.1.7 Event-Exits

Die Event-Exits INPUT, START, SHUT und VORGANG dürfen keine KDCS-Aufrufe enthalten. Sie sind als Unterprogramme zu schreiben und müssen mit der Anweisung *return* beendet werden.

Bei START, SHUT und VORGANG werden die Adressen des Kommunikationsbereichs (KB) und des Standard-Primären-Arbeitsbereichs (SPAB) als Parameter übergeben; dementsprechend müssen diese Bereiche deklariert werden (wie bei den Teilprogrammen mit KDCS-Aufrufen). In "Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen" finden Sie ein Beispiel für einen kombinierten START/SHUT-Exit.

Beim INPUT-Exit übergibt openUTM eine Adresse. Diese Adresse bezeichnet den INPUT-Parameterbereich. Für die Struktur des INPUT-Parameterbereichs steht die Include-Datei *kcinp.h* zur Verfügung; der Name der Datenstruktur ist *kc_inp*. In "Beispiel: INPUT-Exit (BS2000-Systeme)" finden Sie ein Beispiel für einen INPUT-Exit. Auf BS2000-Systemen kann zusätzlich die Adresse eines Steuerfeldbereichs übergeben werden. Für den Steuerfeldbereich steht die Include-Datei *kccf.h* zur Verfügung.

Pro Anwendung darf es jeweils maximal acht Event-Exits START und SHUT geben. INPUT und VORGANG darf es nur jeweils einmal geben. Weitere Informationen zu den Event-Exits finden Sie im Kapitel „Event-Funktionen“.

10.1.8 Programmierung der KDCS-Fehlerbehandlung

An der KDCS-Schnittstelle in C/C++ wird der Returncode *kcrccc* im Rückgabebereich des Kommunikationsbereichs als Character-Feld der Länge drei geliefert. Ohne die C/C++-Makros muss der Wert mittels der *strncmp*-Funktion überprüft werden.

Bei Verwendung der Makroschnittstelle ist die Abfrage einfacher:
Die Include-Datei *kcmac.h* definiert in der static-Variablen *long* KCRCC einen Returncode, der nach jedem Makroaufruf den integer-Wert des *kcrccc*-Felds enthält. Eine Fehlerabfrage beschränkt sich somit auf die Überprüfung von KCRCC.

Zum Vergleich eine Fehlerabfrage nach einem KDCS-Aufruf:

- Bei unmittelbarem Aufruf:

  ```
  if(strncmp(kb->rti.kcrccc,"000",3) != 0) ...
  ```

- Bei Verwendung eines Makros:

  ```
  if (KCRCC != 0) ...
  ```

Um eine bessere Diagnose zu gewährleisten, werden bei Verwendung der Makros alle nicht benutzten Parameter vor einem KDCS-Aufruf automatisch auf binär null gesetzt. Somit enthält der gesamte Parameterbereich einen genau definierten Inhalt. Bei einem Fehler kann der Parameterbereich gezielt überprüft werden, jede Abweichung vom vorgesehenen Inhalt wird entdeckt.

10.1.9 Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme)

Auf Unix- und Linux-Systemen stehen zusätzliche Funktionen zur Verfügung, die über die normale KDCS-Fehlerbehandlung hinausgehen:

- User Signal Routinen

  Dazu stehen Ihnen die beiden Funktionen KCX_REG_USER_SIGNAL_HANDLER() und KCX_UN_REG_USER_SIGNAL_HANDLER () für die Registrierung und De-Registrierung der User Signal Routine zur Verfügung. Die User Signal Routine wird beim Auftreten eines Signals aufgerufen und muss vom Anwender erstellt werden.
- Austausch des utmwork-Prozesses nach PEND RE mit KCX_SET_RELOAD_FLAG()
- Erzeugen eines UTM-Dumps mit KCX_WRITE_DUMP()

Die Prototypen für alle Funktionen befinden sich im ausgelieferten C-Headerfile `kcerrh.h` unter *utmpfad*/`include`.

10.1.9.1 User Signal Routinen (Unix- und Linux-Systeme)

Eine User Signal Routine wird mit der Funktion KCX_REG_USER_SIGNAL_HANDLER() im Teilprogramm oder im Start-Exit registriert, d.h. erst dann ist die anwenderspezifische Signalbehandlung aktiviert.

Mit KCX_UN_REG_USER_SIGNAL_HANDLER() wird sie wieder de-registriert und damit deaktiviert.

Funktion KCX_REG_USER_SIGNAL_HANDLER()

Die Funktion KCX_REG_USER_SIGNAL_HANDLER() benötigt als Aufruf-Parameter einen Funktionspointer, der auf eine User Signal Routine zeigt.

C-Prototyp

```
typedef void utm_user_signal_function_t(int *, char *, char *);
void KCX_REG_USER_SIGNAL_HANDLER(utm_user_signal_function_t
                                 **p2p_user_signal_func_param);
```

Beispiel

```
void my_user_signal_handler(int * p_signal_number,
                            char * signal_text,
                            char * stack_info);
typedef void utm_user_signal_function_t(int *, char *, char *);
static utm_user_signal_function_t * p_user_signal_func;
p_user_signal_func = my_user_signal_handler;
KCX_REG_USER_SIGNAL_HANDLER(&p_user_signal_func);
```

Funktion KCX_UN_REG_USER_SIGNAL_HANDLER()

Durch den Aufruf von KCX_UN_REG_USER_SIGNAL_HANDLER() aus dem Anwendungsprogramm wird die aktuelle User Signal Routine bei openUTM wieder deregistriert, d.h. deaktiviert. Ab diesem Zeitpunkt ist wieder die Standard Fehlerbehandlung von openUTM aktiviert.

C-Prototyp

```
void KCX_UN_REG_USER_SIGNAL_HANDLER(void);
```

Programmierung der Signal Routine

Die User Signal Routine erhält immer drei Aufrufparameter:

Param-1

Die System Signal-Nummer als binäre, 4 Byte lange Variable, d.h. Datentyp "int*".

Param-2

Mit Null-Byte abgeschlossene maximal 255 Bytes lange "abdruckbare" Erklärung der System Signalnummer, z. B. "Segmentation fault" bei Signal 11.

Param-3

Eine mit Null-Byte abgeschlossene, mehrzeilige und maximal 4096 Bytes lange Stack-Information.

Die User Signal Routine sollte nur "Aufräumaktionen" veranlassen und das Programm mit PEND RE beenden:

- KDCS-Aufruf RSET,
- KDCS-Aufruf MPUT NE mit Fehler-Infos aus den Parametern der Anwender-Signal-Routine,
- KDCS-Aufruf PEND RE mit KCRN="Folge-TAC".

Hinweis

Die Funktion darf nicht mit *exit()* verlassen werden.

Beispiel

```
static void my_user_signal_handler( int* p_signal_number
          , char * utm_signal_string
          , char * utm_stack_string )
{
struct kc_ca *pntrCa ;
struct spab *pntrSpab;
fprintf(stderr, "Error Handler \"my_user_signal_handler\"
          called:\n" "signal_number %d\n"
          "utm_signal_string \"%s\"\n" "utm_stack_string \"%s\"\n",
          *p_signal_number, utm_signal_string, utm_stack_string);
fflush(stderr);
```

Danach folgen z.B. KDCS("RSET"), KDCS("MPUT") und KDCS("PENDRE").

Beispiel für an eine User Signal Routine übergebene Parameter

```
utm-signal-number: 8
utm-signal-string: " Arithmetic Exception"
utm-stack-string:
[5] KCSSIGNAL(signal_nbr = 8) (optimized), at 0xfebb39a8 (line ~2458) in
"kcxrtst.c"
[6] __sighndlr(0x8, 0x0, 0xffbfbff8, 0x2f1a4, 0x0, 0x0), at 0xfe1ca79c
[7] fb(), line 115 in "regsignal.c"
[8] fa(), line 104 in "regsignal.c"
[9] regsig(kb = 0xfed3a788, spab = 0xfed42808), line 82 in "regsignal.c"
[10] KDCCC(iutmhlpar = 0xfed4cdd8), line 855 in "root.c"
[11] KDCHLLC(iutmhlpar = 0xfed4cdd8, lgcon = 0x30400) (optimized), at
0xfebb26ac (line ~1922) in "kcxrtst.c"
[12] START_TEILPROGRAM(adfptr01 = 0xffbfc940 "F^A") (optimized), at
0xfec0223c (line ~14697) in "kdcrtmm.c"
[13] KDCRTMM(RTMM_CALL = 0xffbfd0e4 "KDCRTSI \xfe\xd1\xec`") (optimized), at
0xfebf9af4 (line ~9766) in "kdcrtmm.c"
[14] KDCRTSI() (optimized), at 0xfec04dc0 (line ~818) in "kdcrtsi.c"
[15] KDCRTST(argc = 80, argv = <value unavailable>, pKDCMDATA = 0x20)
(optimized), at 0xfebb1f74 (line ~1786) in "kcxrtst.c"
[16] kcxmnt(argc = 9, argv = 0xffbfda74), line 568 in "xirtend.h"
[17] main(argc = 9, argv = 0xffbfda74, envp = 0xffbfda9c) (optimized), at
0x13b1c (line ~64) in "mainutm.c"
```

10.1.9.2 utmwork-Prozess austauschen (Unix- und Linux-Systeme)

Mit der Funktion KCX_SET_RELOAD_FLAG() wird der Austausch eines utmwork-Prozesses nach dem KDCS-Aufruf PEND RE veranlasst.

C-Prototyp

```
void KCX_SET_RELOAD_FLAG(void);
```

Aufruf

```
KCX_SET_RELOAD_FLAG();
```

10.1.9.3 UTM-Dump erstellen (Unix- und Linux-Systeme)

Mit der Funktion KCX_WRITE_DUMP() wird das Erzeugen eines UTM-Dumps veranlasst.

C-Prototyp

```
void KCX_WRITE_DUMP(char * reason);
```

Die Funktion benötigt einen Pointer auf ein 6 Byte langes Feld, das den Grund für den Dump enthält.

Beispiel für den Aufruf

```
KCX_WRITE_DUMP("MYDMP1");
```

10.1.10 Modifizieren von KDCS-Attributen (BS2000-Systeme)

Wenn Sie +Formate oder #Formate verwenden, können Sie die Attribute von Format-Feldern im Programm ändern.

Die KDCS-Attributkombinationen sind in der Include-Datei *kcat.h* enthalten und werden per Include-Anweisung in das Teilprogramm kopiert.
Die Include-Datei *kcat.h* enthält auch das Makro KDCATTR, mit dem Sie die KDCS-Attributkombinationen setzen können. KDCATTR wird wie folgt aufgerufen:

```
KDCATTR (Attributfeld, Attributwert);
```

Die Namen und Eigenschaften der möglichen KDCS-Attributkombinationen sind im Handbuch Ihres Formatierungssystems aufgeführt.

Beispiel

Das Feld "name" soll am Bildschirm als geschütztes Feld ausgegeben werden; dabei ist *a_name* das zugehörige Attributfeld.

```
KDCATTR (spab->std_mask.a_name, KCPROT);
```

Das +Format "FORMAT5" enthält die Felder "FELD1" bis "FELD5". FELD4 soll blinkend an den Bildschirm geschickt werden, alle anderen Felder erhalten die Attribute aus der Formatbeschreibung.

```
#include <kcat.h>
unsigned short mput_features;
     ...
    {...  a_format5 maske_aus; ...} *spab; 1)
     ...
 /* MPUT-Aufruf */
     ...
 KDCATTR (maske_aus.a_feld4, KCSIGN);
 KDCATTR (maske_aus.a_feld1, KCNOATTR);
 KDCATTR (maske_aus.a_feld2, KCNOATTR);
 KDCATTR (maske_aus.a_feld3, KCNOATTR);
 KDCATTR (maske_aus.a_feld5, KCNOATTR);
 ...
 KDCS_MPUTNT (&maske_aus, sizeof(a_format5),
              KDCS_SPACES,KDCS_SPACES,mput_features);
```

[1)] Adressierungshilfe für das +Format "format5"

10.1.11 Plattform-spezifische Besonderheiten auf BS2000-Systemen

Übersetzen von C/C++-Teilprogrammen

Soll bei der Übersetzung von C/C++-Teilprogrammen ein Bindelademodul (LLM) erzeugt werden, das aus einer Code- und einer Daten-CSECT besteht, dann müssen Sie beim Aufruf des Compilers für die COMPILER-ACTION-Option MODULE-GENERATION(MODULE-FORMAT=LLM) setzen. Das LLM muss zum Binden mit dem BINDER in einer PLAM-Bibliothek zur Verfügung gestellt werden.
Für Ihre C-Teilprogramme können auch Objekt-Module erzeugt werden (Typ=R im LMS).

Shareable Code nutzen

Wenn Sie beabsichtigen, C/C++-Programmteile shareable zu laden, müssen Sie bereits beim Übersetzen folgende Option anbeben:

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

Der Shareable Code muss nicht unbedingt in einem eigenen Objektmodul abgelegt werden, sondern kann zusammen mit dem nicht-shareable Teil in einem LLM stehen, der in eine Public und eine Private Slice unterteilt ist.

Die shareable Programmteile brauchen für alle Prozesse (Tasks) der Anwendung(en) gemeinsam nur einmal geladen werden. In den task-lokalen Speicher müssen dann nur noch die nicht-shareable Teile geladen werden.

openUTM bietet verschiedene Möglichkeiten, shareable Objekte zu laden:

- als nicht-privilegiertes Subsystem,
- mit dem ADD-SHARED-PROGRAM-Kommando in den Systemspeicher,
- in einen Common Memory Pool im Benutzerspeicher (Klasse 6-Speicher).

Weitere Informationen zum Übersetzen von Shareable Code finden Sie im Benutzerhandbuch Ihres Compilers. Über das Binden und Laden von Shareable Code informiert ausführlich das openUTM-Handbuch „Anwendungen generieren“ sowie das openUTM-Handbuch „Einsatz von UTM-Anwendungen auf BS2000-Systemen“.

Formaterstellung mit dem IFG

Wie Sie Formate mit dem IFG erstellen können, ist ausführlich im IFG-Handbuch beschrieben. Wenn diese Formate für den Einsatz mit openUTM erstellt werden, so beachten Sie bitte folgende Hinweise:

- Der Formatname darf höchstens 7 Zeichen lang sein.
- Im Benutzerprofil wählen Sie die "Struktur des Datenübergabebereichs":
  - für #Formate: getrennte Attributblöcke und Feldinhalte
  - für *Formate: nicht ausgerichtet, ohne Attributfelder
  - für +Formate: nicht ausgerichtet, mit Attributfeldern
- Für die Programmiersprachen C/C++ erzeugt der IFG immer **eine** Adressierungshilfe. Felder mit dem Attribut "Felddatentyp arithmetisch" werden auf Zeichenketten (char [..]) abgebildet.

- Bitte beachten Sie bei der Definition der Adressierungshilfen für *Formate und +Formate, dass openUTM beim MGET bzw. FGET den Transaktionscode aus der Nachricht entfernt (sofern dies nicht in einem INPUT-Exit explizit verhindert wird). Wenn das erste Feld im Format den Transaktionscode enthält, so können Sie dies berücksichtigen, indem Sie die Nachricht in das zweite Feld einlesen.

Beispiel

```
struct work
   { union kc_paa param;
     FORM1 std_mask;               /* Adresseriungshilfe fuer das*/
          .                        /* *Format FORM1 daklarieren  */
    } *spab;
          .
    /* MGET-Aufruf */
     KDCS_MGET ( spab->std_mask.FUNCTION               1)
                ,sizeof( FORM1 )
                ,"*FORM1  "               );
          .
    /* MPUT-Aufruf */
     KDCS_MPUTNT (&spab->std_mask,sizeof(FORM1)
                 ,KDCS_SPACES,"*FORM1  ",KCNODF);
    /* PEND FI-Aufruf */
           .
```

[1] FUNCTION ist das zweite Eingabefeld des Formates.

- Bei der Einsatzvorbereitung bringen Sie die Formate in die Formateinsatzdatei (Formatbibliothek). Diesen Namen geben Sie bei den FHS-Startparametern an.

Erweiterter Zeilenmodus

Für die Arbeit im erweiterten Zeilenmodus müssen Sie die Steuerzeichen, sofern diese nicht in der Sprache C/C++ zur Verfügung stehen, selbst definieren. Welche Steuerzeichen welchen hexadezimalen Werten entsprechen, finden Sie im „TIAM Benutzerhandbuch“, z.B. bei der Beschreibung des Makros VTCSET.

10.1.12 Plattform-spezifische Besonderheiten auf Unix- und Linux-Systemen

Signalbehandlung

Auf Unix- und Linux-Systemen ist in den C-Teilprogrammen eine eingeschränkte Nutzung von Signalen möglich. Im Startexit eines Workprozesses können die Signale SIGUSR1 und SIGUSR2 vom C-Teilprogramm gefangen werden. Alle anderen Signale werden vom openUTM Systemcode standardmäßig selbst behandelt.

Alternativ zur standardmäßigen Behandlung von Signalen in openUTM können Sie für bestimmte Signale eine eigene User Signal Routine programmieren und bei openUTM registrieren, siehe Abschnitt „Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme)“.

CMX Schnittstelle in utmwork

Bei der Verwendung von CMX-Aufrufen in C/C++ Teilprogrammen müssen Sie Folgendes beachten:

In einer UTM-Anwendung mit OSI TP wird aus technischen Gründen eine CMX-Simulation in der UTM Bibliothek libwork.a bzw. libwork.so verwendet. Deshalb ist es in diesem Fall nicht möglich in einem C/C++ Teilprogramm CMX-Aufrufe zu verwenden.

Aufruf der Funktion fork() in einem C/C++-Teilprogramm

Bei Aufrufen der Funktion `fork()` in C/C++-Teilprogrammen müssen Sie Folgendes beachten: In einem durch `fork()` erzeugten Kind-Prozess dürfen keine Programmschnittstellen von openUTM verwendet werden. Anderenfalls beendet sich die UTM-Anwendung abnormal.

Einzelheiten zum Erstellen und Binden von Shared Objects, zum Übersetzen von Teilprogrammen und zum Binden einer Anwendung finden Sie im Handbuch openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“ im Kapitel „Anwendungsprogramm erzeugen.

10.1.13 Plattform-spezifische Besonderheiten auf Windows-Systemen

Auf Windows-Systemen müssen die Projekte mit Visual Studio ab Version 2010 erzeugt werden.

Auf Windows-Systemen werden keine Signale unterstützt.

Einzelheiten zum Erstellen von DLLs, zum Übersetzen von Teilprogrammen und zum Binden einer Anwendung mit Visual Studio finden Sie im Handbuch openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“ im Kapitel „Anwendungsprogramm erzeugen“.

10.2 Programmierbeispiele in C/C++

In diesem Abschnitt finden Sie Beispiele zur Codierung einzelner KDCS-Aufrufe über die C/C++-Makroschnittstelle, Beispiele für ein vollständiges C-Programm, einen INPUT-Exit, einen Event-Service MSGTAC sowie ein Beispiel für eine komplette UTM-Anwendung.

10.2.1 Beispiele zu einzelnen KDCS-Aufrufen

In diesem Abschnitt finden Sie Codierbeispiele für folgende KDCS-Aufrufe:

- MGET
- MPUT
- DPUT
- MCOM mit DPUT im Auftrags-Komplex
- APRO mit MPUT bei verteilter Verarbeitung

Da die übrigen KDCS-Aufrufe auf analoge Weise codiert werden, wird an dieser Stelle auf eine explizite Darstellung aller Aufrufe verzichtet.

Beim KDCS-Aufruf bezeichnet *&pa* die Adresse des KDCS-Parameterbereichs und *&ma* die Adresse des Nachrichtenbereichs.

MGET-Aufruf

- Eine unformatierte Dialog-Nachricht von 80 Bytes Länge soll empfangen werden. Wurde sie irrtümlicherweise kürzer gesendet, soll eine erneute Eingabe angefordert werden.

```
KDCS_MGET (&ma,80,KDCS_SPACES);
if (KCRCC != 0)
   mget_error();
if ( pa.kcla > ca->ca_return.kcrlm)
    r_mput ();
```

  In der Routine *r_mput ()* wird mit MPUT eine Aufforderung zur Eingabe-Wiederholung an das Terminal gesendet.

- In einem laufenden Vorgang kann eine Eingabe kommen, die aus einer Kurznachricht, erzeugt mit der Funktionstaste F2, sowie aus Daten von 10 Zeichen besteht. Sie soll eine Sonderfunktion auslösen. Der Taste F2 wurde beim Generieren der Returncode 21Z zugewiesen.

```
KDCS_MGET (&ma,input_lth,dev.features);
if (KCRCC == 21 ) {                         /* Return-Code for F2     */
  KDCS_MGET (&ma,10,KDCS_SPACES);
  if (KCRCC != 0 )
     mget_error();
  }
```

- *BS2000-Systeme:*
  Das Format "FORM15" wurde von einem Terminal angefordert. Die Länge der ungeschützten Daten beträgt 500 Zeichen in verschiedenen Formatfeldern. Dieses Format soll im Programm empfangen werden. FORM15 wurde im Programm als `std_mask` deklariert.

```
KDCS_MGET (&ma.std_mask,500,"*FORM15 ");
if (KCRCC == 5)                          /* Invalid Format-ID      */
   format_error();
if (KCRCC != 0)
   mget_error();
```

  In der Routine format_error muss das Format nochmals ausgegeben werden, um mit dem richtigen Format weiterarbeiten zu können.

MPUT-Aufruf

- Eine unformatierte Nachricht von 80 Bytes soll an das Terminal gesendet werden.

```
KDCS_MPUTNE (&ma,80,KDCS_SPACES,KDCS_SPACES,KCNODF);
if (KCRCC != 0 )
   mput_error();
```

- *BS2000-Systeme:*
  Die letzte Nachricht in einem Vorgang soll an ein Terminal im Format-Modus geschickt werden. Der Name des *Formats ist "FORM15". Der Bildschirm soll vorher gelöscht werden.

```
KDCS_MPUTNE (&ma,500,KDCS_SPACES,"*FORM15 ",KCREPL);
if (KCRCC != 0 )
   mput_error();
```

  REPLACE wird bei Formatwechsel standardmäßig ausgeführt. Die Ausgabe erfolgt, um Fehler wegen undefinierter Feldinhalte auszuschließen.

- *BS2000-Systeme:*
  In einem *Format "FORM10", das laut letzter Eingabe am Terminal noch vorhanden ist, sollen als Antwort alle ungeschützten Felder gelöscht werden.

```
KDCS_MPUTNE (&ma,0,KDCS_SPACES,"*FORM10 ",KCERAS);
if (KCRCC != 0 )
   mput_error();
```

DPUT-Aufruf

- Ein Asynchron-Auftrag mit einer Nachricht von 11 Zeichen soll am 11.11. (= 315. Tag im Jahr) um 11.11 Uhr an ein Teilprogramm gegeben werden (absolute Zeitangabe). Der TAC lautet "ALAAF".

```
KDCS_DPUTNE (&ma,11,"ALAAF   ",KDCS_SPACES,0,'A',315,11,11,0);
if (KCRCC != 0 )                         /* A = absolute time      */
   dput_error();
```

- *BS2000-Systeme:*
  Eine Asynchron-Nachricht von 80 Zeichen soll nach 1 Stunde an das Terminal *DSS1* ausgegeben werden (relative Zeitangabe). Dabei soll die Bildschirmfunktion *akustischer Alarm (BEL)* ausgelöst werden.

```
KDCS_DPUTNE (&ma,80,"DSS1    ",KDCS_SPACES,KCALARM,'R',0,1,0,0);
if (KCRCC != 0 )                          /* R = relative time      */
   dput_error();
```

Auftrags-Komplex: MCOM- und DPUT-Aufruf

Eine formatierte Asynchron-Nachricht (200 Byte) soll am selben Tag um 18.00 Uhr an den Drucker PRINTER2 gesendet werden. Die Quittung vom Drucker wird per Programm behandelt.

Bei positiver Quittung erhält ein Asynchron-Programm mit dem TAC PRINTPOS einen Quittungsauftrag mit einer Nachricht in der Länge von 20 Byte, bei negativer Quittung wird ein Asynchron-Programm mit dem TAC PRINTNEG gestartet (ohne Nachricht). Zur negativen Quittung wird eine Benutzerinformation in der Länge von 80 Byte protokolliert; diese kann mit DADM UI gelesen werden.

Der Auftrags-Komplex wird durch zwei MCOM-Aufrufe eingerahmt; dabei werden die Ziele von Druckauftrag (=Basisauftrag) und Quittungsaufträgen im Aufruf MCOM BC festgelegt; die Komplexidentifikation lautet "*PRICOMP".

```
/* Begin of the complex                                                 */
   KDCS_MCOMBC ("PRINTER2","PRINTPOS","PRINTNEG","*PRICOMP");
   if (KCRCC != 0 )
      mcom_error();
/* DPUT-message for printer                                             */
   KDCS_DPUTNE (&ma1,200,"*PRICOMP","*FORM1   ",KCNODF,'A',
                ca->ca_head->kccv_doy,18,0,0);
   if (KCRCC != 0 )
      dput_error();
/* acknowledgement job in positive case                                 */
   KDCS_DPUTPT (&ma2,20,"*PRICOMP");
   if (KCRCC != 0 )
      dput_error();
/* User-information in negative case                                    */
   KDCS_DPUTMI (&ma3,80,"*PRICOMP");
   if (KCRCC != 0 )
      dput_error();
/* acknowledgement job in negative case                                 */
   KDCS_DPUTMT (&ma2,0,"*PRICOMP");
   if (KCRCC != 0 )
      dput_error();
/* End of complex                                                       */
   KDCS_MCOMEC ("*PRICOMP");
   if (KCRCC != 0 )
      mcom_error();
```

Verteilte Verarbeitung: APRO-Aufruf mit anschließendem MPUT

Vom Auftraggeber-Vorgang aus soll der Dialog-Vorgang mit dem Transaktionscode *LTAC1* der Anwendung *PARTNER1* adressiert werden (zweistufige Adressierung). Dabei soll dem Auftragnehmer-Vorgang die Vorgangs-Identifikation *>VGID1* zugeordnet werden. Anschließend wird eine MPUT-Nachricht mit Länge 100 im Zeilenmodus an die Partner-Anwendung geschickt.

```
KDCS_APRODM ("LTAC1   ","PARTNER1",">VGID1  ");
if (KCRCC != 0 )
   apro_error();
...
KDCS_MPUTNE (&ma,100,">VGID1  ",KDCS_SPACES,KCNODF);
if (KCRCC != 0 )
   mput_error();
```

10.2.2 Beispiel für ein vollständiges C-Teilprogramm

Beispiel eines ablauffähigen KDCS-Teilprogramms in der Programmiersprache C.
Das Programm gibt den Text "hello world !" und den Namen des logischen Terminals aus.

Programm mhello

```
#include <kcmac.h>
struct work {
       union  kc_paa call_pb;
       char   buffer[400];
      };
struct kc_ca {
       struct ca_hdr kopf;
       struct ca_rti rfld;
       char   kcprg[500];
      };
#define NB     spab->buffer
#define KBKOPF kb->kopf
#define KBRFLD kb->rfld
#define PB     spab->call_pb
void mhello ( struct kc_ca *kb, struct work *spab )
{
   /* KDCS interface initialisation */
   KDCS_SET( &PB, &KBKOPF, &KBRFLD );
   /* INIT - Call */
   KDCS_INIT( sizeof(struct kb->kcprg), sizeof(struct work) );
   /* MPUT-NT - Call   */
   strcpy ( NB, "hello world !\n\n" );
   KDCS_MPUTNT( NB, (short)strlen(NB), KDCS_SPACES,
                KDCS_SPACES, KCNODF );
   /* MPUT-NT - Call   */
   sprintf ( NB, "lterm = %.8s \n", KBKOPF.kclogter );
   KDCS_MPUTNT( NB, (short)strlen(NB), KDCS_SPACES,
                KDCS_SPACES, KCNODF );
   /* PEND-FI - Call   */
   KDCS_PENDFI();
}
```

10.2.3 Beispiel: Event-Exit INPUT (BS2000-Systeme)

Der INPUT-Exit *forinput* wird bei Eingaben im Formatmodus aufgerufen und reagiert auf die Eingaben wie folgt:

Benutzerkommandos werden abgesetzt:

- KDCOUT: Drücken der Taste F1
- KDCDISP: Drücken der Taste F2
- KDCOFF: erstes Zeichen der Eingabe ist "/"; wird nur außerhalb eines Vorgangs akzeptiert.

Soll dem Benutzer zusätzlich die Eingabe von KDCLAST und KDCFOR erlaubt werden, muss das Programm entsprechend erweitert werden.

Dieser INPUT-Exit wird mit der KDCDEF-Anweisung EXIT generiert:

KDCDEF-Anweisung

```
EXIT PROGRAM=FORINPUT,USAGE=(INPUT,FORMMODE)
```

Event-Exit INPUT

```
#include <string.h>
#include <kcinp.h>
#define KDCDISP 2
#define KDCOUT  1
#define NOKEY 0
#define FKEY 1
#define KDCOFF "KDCOFF  "
#define NOTAC (strncmp (param->kcicfinf, "ON", 2) != 0)
#define CV_END (strncmp (param->kcicvst, "EC", 2) == 0)
#define TAC       param->kcicvtac[0]
#define fkey      param->kcifkey
#define cmd       param->kcintac
#define nexttac   param->kcintac
#define errcode   param->kcierrcd
#define contcode  param->kciccd
#define firstchar param->kcifch[0]
#define cut       param->kcicut
static int  key( struct kc_inp * );
static void func_control( struct kc_inp * );
static void cv_continue( struct kc_inp * );
void forinput ( struct kc_inp *param )
{
  if ( key ( param ) == NOKEY)                    /* No F-Key   */
     { if ( CV_END)
          func_control ( param );
       else
          cv_continue ( param );
     }
}

/*********************************************************************/
/*               function key for checking F-key                     */
/*********************************************************************/
int key ( struct kc_inp *param )
{
```

```
  int key_value = NOKEY;
  if (fkey > 0)
     {  switch (fkey)
         {
          case KDCOUT:
              memcpy (cmd, "KDCOUT  ", 8);
              memcpy (contcode, "CD", 2);
              cut =  N ;
              memset (errcode,    , 4);
              key_value = FKEY;
              break;
          default: break;
         }
     }
  return key_value;
}
/*************************************************************************/
/*   function func_control: checking the next FUNCTION out of conversation*/
/************************************************************************* */
void func_control ( struct kc_inp * param )
{
  if (firstchar ==  / )                      /* check the first character*/
     { memcpy (cmd, KDCOFF, 8);              /* of input:   /  = KDCOFF  */
       memcpy (contcode, "CD", 2);
       cut =  N ;
       memset (errcode,    , 4);
     }
  else                                       /* check control field      */
     {
       if (NOTAC)                            /* no input in control field*/
          { memset (nexttac,    , 8);
            memcpy (contcode, "ER", 2);
            cut =  N ;
            memcpy (errcode, "ER01", 4);
          }
       else
          { switch (TAC)                     /* TAC for the next         */
             {                               /* conversation             */
              case  1 :
                  memcpy (nexttac, "DTAC1   ", 8);
                  memcpy (contcode, "SC", 2);
                  cut =  Y ;
                  memset (errcode,    , 4);
                  break;
              case  2 :
                  memcpy (nexttac, "DTAC3   ", 8);
                  memcpy (contcode, "SC", 2);
                  cut =  Y ;
                  memset (errcode,    , 4);
                  break;
              case  3 :
                  memcpy (nexttac, "DTAC6   ", 8);
                  memcpy (contcode, "SC", 2);
                  cut =  Y ;
                  memset (errcode,    , 4);
                  break;
              default:                       /* TAC is invalid           */
                  memset (nexttac,    , 8);
                  memcpy (contcode, "ER", 2);
```

```
                cut =  N ;
                memcpy (errcode, "ER02", 4);
                break;
             }
          }
     }
}
/*****************************************************************************/
/*          function cv_continue: continue the conversation              */
/*****************************************************************************/
void  cv_continue ( struct kc_inp * param )
{  memset (nexttac,    , 8);
   memcpy (contcode, "CC", 2);
   cut =  N ;
   memset (errcode,    , 4);
}
```

10.2.4 Beispiel: Event-Service MSGTAC

Der MSGTAC-Event-Service NOHACK zählt die Anzahl der Fehlversuche in einem TLS. Wenn openUTM ein KDCSIGN akzeptiert (d.h. Meldung K008 oder K033), so wird dieser TLS wieder gelöscht.

Falls nach drei ungültigen KDCSIGN-Versuchen der 4. KDCSIGN-Versuch wieder fehlerhaft ist, so soll das entsprechende Terminal automatisch diskonnektiert werden, und zwar per FPUT-Aufruf mit KCRN="KDCPTRMA". Der Nachrichtenbereich enthält die Parameter des Administrationskommando KDCPTRMA, siehe auch openUTM-Handbuch „Anwendungen administrieren“:

```
PTERM=pterm,PRO=proname,ACT=DIS
```

Die K-Meldungen werden jeweils mit FGET vom MSGTAC-Teilprogramm gelesen. Nach der "Verarbeitung" einer K-Meldung wird mit FGET sofort die nächste K-Meldung gelesen, innerhalb desselben Teilprogrammlaufs.

Event-Service MSGTAC

```
#include <stdio.h>
#include <kcmac.h>
#include <kcmsg.h>
#define _K008 (memcmp (NR, "K008", 4) == 0)
#define _K033 (memcmp (NR, "K033", 4) == 0)
/* K008: KDCSIGN accepted                                          */
/* K033: Start-Format                                              */
#define MESSAGE_OK (_K008 || _K033)
#define _K004 (memcmp (NR, "K004", 4) == 0)
#define _K006 (memcmp (NR, "K006", 4) == 0)
#define _K031 (memcmp (NR, "K031", 4) == 0)
/* K004: Invalid Identification                                    */
/* K006: Invalid password                                          */
/* K031: Card not ok                                               */
#define OTHER_MESSAGE !(_K004 || _K006 || _K008 || _K031 || _K033)
#define HACK_MAX 3
#define PTERM " PTERM="
#define PRONAM ",PRONAM="
#define DIS ",ACTION=DIS"
#define OFF ",STATUS=OFF"
#define kcrc_ca->ca return.kcrccc
#define pa spab->param
#define NR spab->ma.kcmsgs.msghdr.MSGNR
#define MSG spab->ma.kcmsgs.msg
#define LTERM spab->ma.lterm
#define hacknr spab->hack_nr
#define admin spab->ma.adm

  struct adm_line
      { char pterm_t[6];
        char pterm[8];
        char pronam_t[8];
        char pronam[8];
        char dis_t[11];
        char off_t[11];
      };
  struct ca_area
    { struct ca_hdr ca_head;
      struct ca_rti ca_return;
    };
  struct work
```

```
    { struct kc_pa param;
      short hack_nr;
      struct msg_area
         { char lterm[8];
           struct adm_line adm;
           struct KCMSGS kcmsgs;
         } ma;
      char  buffer[100];
    };
static void set_lterm( struct work * );
static void set_pterm( struct work * );
void NOHACK (struct ca_area *ca, struct work *spab )
{
  int other_message = 0;
/* INIT-Operation                                                    */
KDCS_SET (&spab->param, &ca->ca_head, &ca->ca_return);
KDCS_INIT (0,512);
/*********************************************************************/
/* while-loop: reading and processing all messages                   */
/*********************************************************************/
  while (KCRCC == 0 )
   {
/* FGET-Operation: reading the message                               */
   KDCS_FGET (&spab->ma.kcmsgs,132,KDCS_SPACES);
   if (KCRCC != 0 )
      break;
     if (OTHER_MESSAGE)
       { other_message = 1;
         break;
       }
     set_lterm ( spab );
/* read TLSB                                                    */
     KDCS_GTDA (&hacknr,2,"TLSB",LTERM);
     if (KCRCC != 0 )
        break;
       if ((hacknr < 0) || (hacknr > HACK_MAX))
          hacknr = 0;                       /* Initialize TLS              */
  /* If KDCSIGN is correct, initialize the TLS, if not, count the number   */
  /* of failed attempts. After the fourth invalid KDCSIGN disconnect the   */
  /* correspondending terminal.                                            */
       if ((hacknr < HACK_MAX) && MESSAGE_OK)
          hacknr = 0;                           /* Initialize TLS          */
       else                                     /* invalid KDCSIGN         */
         {
           if (hacknr < HACK_MAX)
              ++hacknr;
           else
             { memcpy (admin.pterm_t, PTERM, 7);
               memcpy (admin.pronam_t, PRONAM, 8);
               set_pterm ( spab );
  /* Disconnect the terminal by asynchronous administration                */
               memcpy (admin.dis_t, DIS, 11);
               memcpy (admin.off_t, OFF, 11);
     KDCS_FPUTNE (&admin,sizeof(struct adm_line),"KDCPTRMA",KDCS_SPACES,KCNODF)
;
     if (KCRCC != 0 )
        break;
               hacknr = 0;
  /* log on User logging                                                   */
```

```
     KDCS_LPUT (&admin,sizeof(struct adm_line));
     if (KCRCC != 0 )
        break;
             }
        }
  /* set up TLSB                                                    */
     KDCS_PTDA (&hacknr,2,"TLSB",LTERM);
     }
  /* ********************************************************************/
  /*                         End of while loop                           */
  /********************************************************************/
   if ( KCRCC != 10  ||  other_message)
  /* other message or error in the while loop                        */
     {
  /* error line                                                      */
     sprintf(spab->buffer, "Error in program unit - conversation  %8.8s"\
                           ", TAC:  %8.8s  because %4.4s. RC= %3.3s " ,
                           ca->ca_head.kccv_tac , ca->ca_head.kcpr_tac ,
                           pa.kcop , KDCS_ERR  );
/* RSET-Operation                                                  */
     KDCS_RSET();
/* LPUT-Operation: log on user logging                             */
     KDCS_LPUT( spab->buffer , strlen( spab->buffer ) );
   }
/* PEND FI-Operation                                               */
 KDCS_PENDFI();
  }
/*********************************************************************/
/*                       function set_lterm ()                        */
/*********************************************************************/
void set_lterm ( struct work * spab )
  { if _K004
       { memcpy (LTERM,MSG.K004.LTRM, 8);
         return;
       }
    if _K006
       { memcpy (LTERM, MSG.K006.LTRM, 8);
         return;
       }
    if _K008
       { memcpy (LTERM, MSG.K008.LTRM, 8);
         return;
       }
    if _K031
       { memcpy (LTERM, MSG.K031.LTRM, 8);
         return;
       }
    if _K033
       { memcpy (LTERM, MSG.K033.LTRM, 8);
         return;
       }
   }
/*********************************************************************/
/*                       function set_pterm ()                        */
/*********************************************************************/
void set_pterm ( struct work *spab )
  { if _K004
       { memcpy (admin.pterm, MSG.K004.PTRM, 8);
         memcpy (admin.pronam, MSG.K004.PRNM, 8);
```

```
         return;
       }
    if _K006
       { memcpy (admin.pterm, MSG.K006.PTRM, 8);
         memcpy (admin.pronam, MSG.K006.PRNM, 8);
        return;
      }
   if _K031
      { memcpy (admin.pterm, MSG.K031.PTRM, 8);
        memcpy (admin.pronam, MSG.K031.PRNM, 8);
        return;
      }
}
```

Das obige Beispiel des MSGTAC-Event-Service zeigt lediglich die Möglichkeiten auf, wie Meldungen geeignet ausgewertet und die Anwendung administriert werden kann.

Zur Überwachung von Sicherheitsverletzungen sollte jedoch die K094-Meldung genutzt werden (SIGNON SILENT-ALARM), da hier auch UPIC- und OSI TP-Clients eingeschlossen sind. Außerdem kann die UTM-Anwendung umfassender über die programmierte Administration (ADMI-Schnittstelle) administriert werden.

10.2.5 Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen

Mit diesem Anwendungsbeispiel für eine UTM-Anwendung auf einem BS2000-System können Adressdaten verwaltet werden, die in einer Datei stehen. Die Anwendung stellt dazu die nachfolgenden Funktionen zur Verfügung, die durch Eintrag des jeweiligen TACs in das dafür vorgesehene Feld aufgerufen werden. Die Ein- und Ausgaben erfolgen in einem Format.

> ¡ Beispiele für eine UTM-Anwendung auf Unix-, Linux- oder Windows-Systemen entnehmen Sie bitte der Beispiel-Anwendung (Unix- und Linux-Systeme) und dem QuickStartKit (Windows-Systeme), die zusammen mit openUTM ausgeliefert werden.

TAC	Funktion	Erklärung
1	Anzeige	gibt eine in der Datei vorhandene Adresse aus. Suchbegriff ist dabei der Name und die ersten zwei Buchstaben des Vornamens, welche in den zugehörigen Feldern anzugeben sind.
2	Neueintrag	trägt eine neue Adresse in die Datei ein. Eine Adresse mit dem gleichen Suchbegriff (s.o.) darf dort nicht schon vorhanden sein.
3	Ändern	ändert einen Adresseintrag. Die Adresse muss in der Datei schon vorhanden sein.
4	Löschen	Löscht eine in der Datei vorhandene Adresse.

Bei Fehlbedienung erscheint in der untersten Zeile des Formats eine Fehlermeldung.

Die oben genannten Ziffern sind die Transaktionscodes (TACs), die die Anwendung steuern. Dabei rufen der Transaktionscode 1 das Teilprogramm "TPREAD" und die Transaktionscodes 2, 3 und 4 das Teilprogramm "TPUPDATE" auf. Diese Teilprogramme verzweigen dann jeweils in das Teilprogramm "TPFILE". Dieses Teilprogramm wird als START- und SHUT-Exit eingesetzt und enthält die Unterprogramme, die die Ein-/Ausgaben auf die Adressdatei durchführen.
Das Teilprogramm "BADTACS" wird vom openUTM automatisch aufgerufen, wenn ein ungültiger TAC eingegeben wird.

Die Funktion "ERRCHECK" behandelt Fehler, die in den Teilprogrammen auftreten.

Nach dem Aufbau der Verbindung mit der Anwendung und erfolgtem KDCSIGN wird sofort von openUTM das Format ausgegeben (Startformat). Die Arbeit mit dem Benutzer erfolgt dann im strengen Dialog, d.h. auf die Eingabe eines TACs und des Schlüssels reagiert die Anwendung mit der Ausgabe des Formats das die gesuchte Adresse enthält bzw. mit einer Erfolgs- oder einer Fehlermeldung in der untersten Zeile.

> ¡ Dieses Programm ist nur gedacht, um zu zeigen wie man mit openUTM programmiert. Die Dateizugriffe sind nicht über das UTM-Transaktionskonzept gesichert.

Die folgenden Struktogramme zeigen den Aufbau der Teilprogramme:

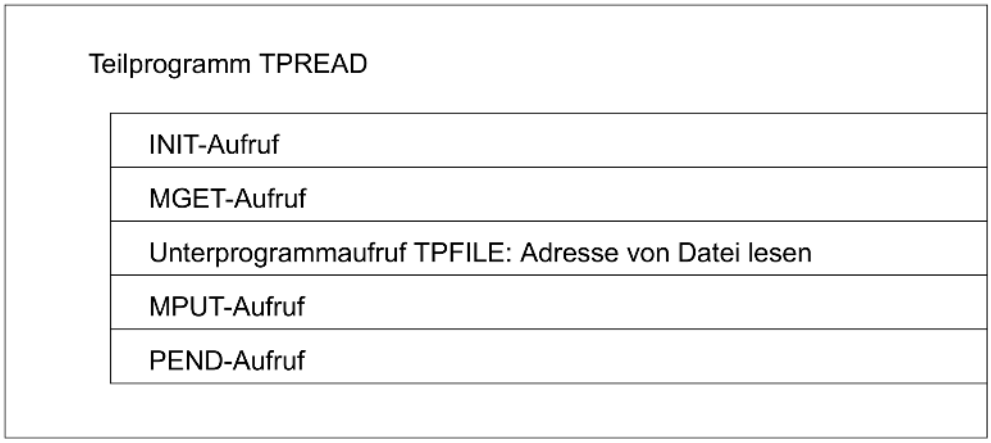


Bild: Struktogramm des Teilprogramms TPREAD

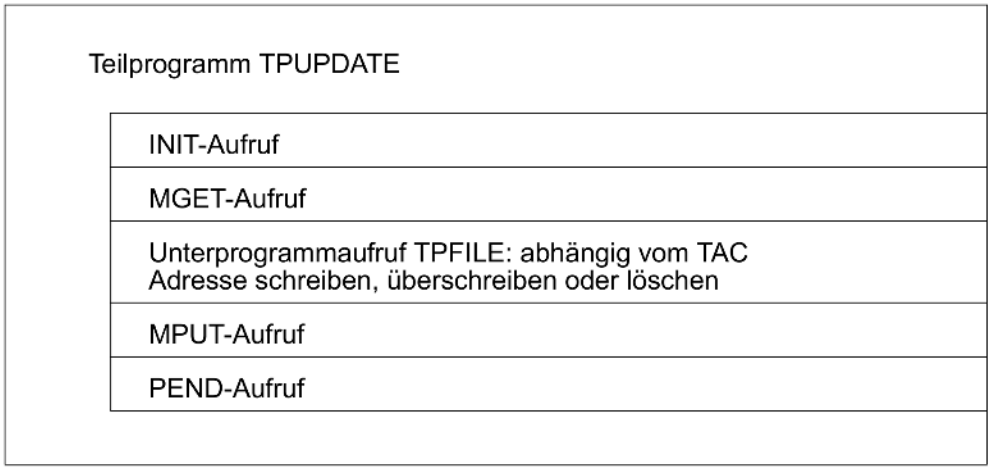


Bild: Struktogramm des Teilprogramms TPUPDATE

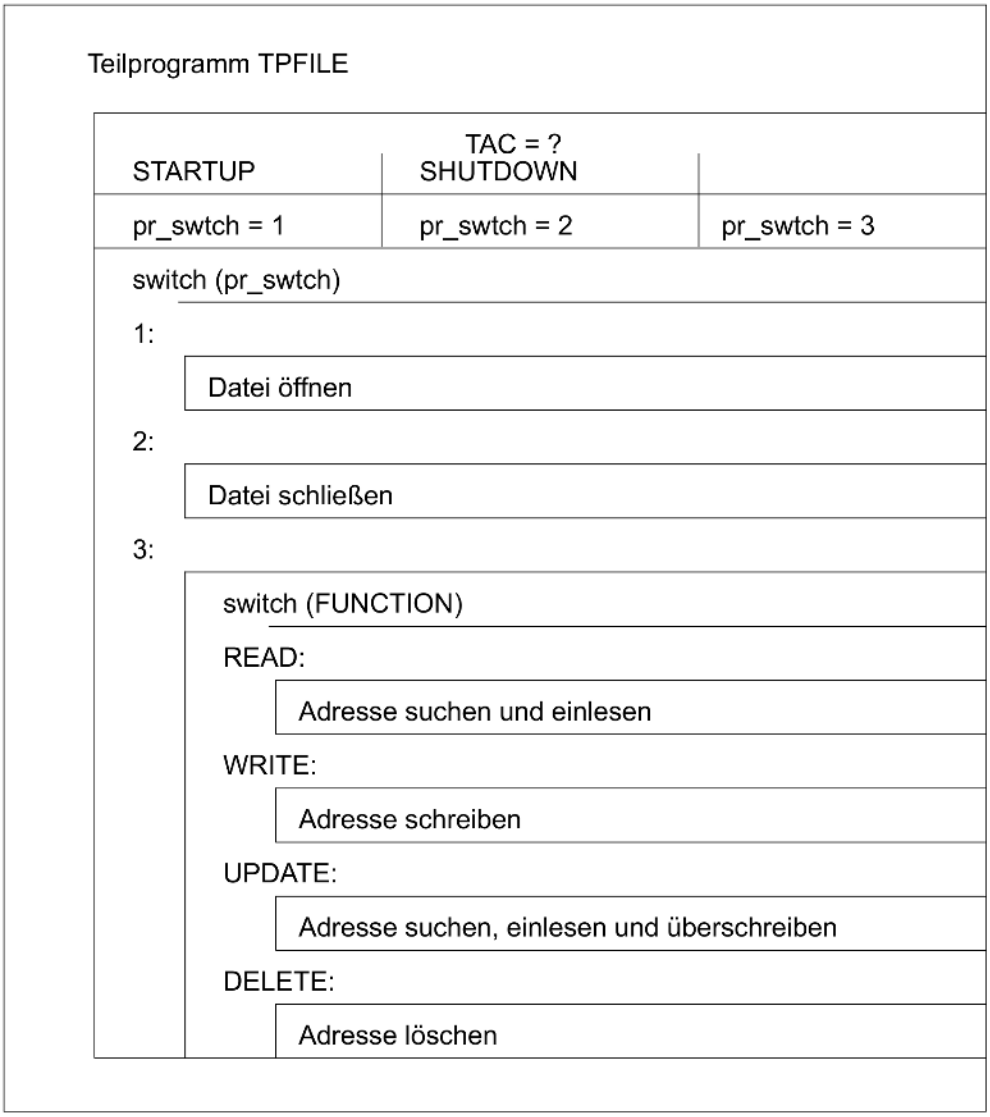


Bild: Struktogramm des Teilprogramms TPFILE

Der Vollständigkeit halber ist im Anschluss an die C-Programme die Generierung dieser Anwendung aufgeführt. Die genaue Bedeutung der einzelnen Operanden und Anweisungen entnehmen Sie bitte dem openUTM-Handbuch „Anwendungen generieren“.

Das folgende Bild zeigt das Format FORM1, das bei dieser Anwendung verwendet wurde:

Format FORM1

```
*******************************************************************************
                A d d r e s s   A d m i n i s t r a t i o n
*******************************************************************************
                            Select function:
-------------------------------------------------------------------------------
      Actual function:
      Name:                                 First Name:
      Street:                               No.:
      Postal Code:                          Residence:
      Phone:
-------------------------------------------------------------------------------
                             Function selection
       1 = Show address               |   4 = Delete address
       2 = Enter address              |
       3 = Update address             |       Finish with  kdcoff
-------------------------------------------------------------------------------
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Im Folgenden ist die Struktur der Adressierungshilfe für dieses Format abgedruckt.

Das Feld FUNCTION ist ein geschütztes Ausgabefeld mit dem Attribut "automatische Eingabe", in das Feld POSTAL_CODE sind nur numerische Eingaben erlaubt und MSGTEXT ist ein geschütztes Ausgabefeld. Die komplette Attributliste können Sie sich ausdrucken lassen. Nähere Informationen hierzu finden Sie im Handbuch Ihres Formatierungssystems.

Include-Datei FORM1.h

```
/* FORMAT NAME      : FORM1    */
/* USER AREA LENGTH :   235    */
   typedef struct {
           char NAME                          [  14 ] ;
           char FIRST_NAME                    [  20 ] ;
           char STREET                        [  26 ] ;
           char NUMBER                        [  10 ] ;
           char POSTAL_CODE                   [   5 ] ;
           char RESIDENCE                     [  24 ] ;
           char PHONE                         [  21 ] ;
   } ADDRESS;
   typedef struct {
           char TAC                           [   8 ] ;
           char FUNCTION                      [  27 ] ;
           ADDRESS addr;
           char MSGTEXT                       [  80 ] ;
   } FORM1 ;
```

Include-Datei tp.h

```
#ifndef TP_H
#define TP_H
#include "FORM1.h"
  #define kcrc ca->ca_return.kcrccc
  #define pa spab->param
    struct ca_area
      { struct ca_hdr ca_head;
        struct ca_rti ca_return;
      };
    struct work
      { union kc_paa param;
        FORM1 std_mask;                       /* area for addressing aid  */
        char progname[8];
       };
  void BADTACS  ( struct ca_area * , struct work * );
  void errcheck ( struct ca_area * , struct work * );
  void TPFILE   ( struct ca_area * , struct work * );
  void TPREAD   ( struct ca_area * , struct work * );
  void TPUPDATE ( struct ca_area * , struct work * );
#endif
```

Teilprogramm TPREAD

```
#include <kcmac.h>
#include "tp.h"
void TPREAD (struct ca_area *ca , struct work *spab )
{
/* INIT-Operation                                                    */
   KDCS_SET (&spab->param, &ca->ca_head, &ca->ca_return);
   KDCS_INIT (0,sizeof(struct work));
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPREAD  ", 8);
         errcheck (ca, spab);
       }
    else
       memcpy (spab->std_mask.TAC, ca->ca_head.kcpr_tac, 8);
/* MGET-Operation                                                    */
   KDCS_MGET ( spab->std_mask.FUNCTION
               ,sizeof( FORM1 )
               ,"*FORM1  "               );
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPREAD", 8);
         errcheck (ca, spab);
       }
/* call function "tpfile" for reading address                       */
  TPFILE (ca, spab);
/* MPUT-Operation                                                    */
   KDCS_MPUTNT (&spab->std_mask,sizeof(FORM1)
                ,KDCS_SPACES,"*FORM1  ",KCNODF);
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPREAD", 8);
         errcheck (ca, spab);
       }
/* PEND FI-Operation                                                 */
  KDCS_PENDFI();
}
```

Teilprogramm TPUPDATE

```
#include <kcmac.h>
#include "tp.h"
void TPUPDATE ( struct ca_area *ca , struct work *spab )
{
/* INIT-Operation                                                    */
   KDCS_SET (&spab->param, &ca->ca_head, &ca->ca_return);
   KDCS_INIT (0,sizeof(struct work));
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPUPDATE", 8);
         errcheck (ca, spab);
       }
    else
       memcpy (spab->std_mask.TAC, ca->ca_head.kcpr_tac, 8);
/* MGET-Operation                                                    */
   KDCS_MGET ( spab->std_mask.FUNCTION
               ,sizeof( FORM1 )
               ,"*FORM1  "               );
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPUPDATE", 8);
         errcheck (ca, spab);
       }
/* call function "tpfile" for updating address                      */
  TPFILE (ca, spab);
/* MPUT-Operation                                                    */
   KDCS_MPUTNT (&spab->std_mask,
                sizeof(FORM1),KDCS_SPACES,"*FORM1  ",KCNODF);
   if (KCRCC != 0 )
       { memcpy (spab->progname, "TPUPDATE", 8);
         errcheck (ca, spab);
       }
/* PEND FI-Operation                                                 */
  KDCS_PENDFI();
}
```

Teilprogramm BADTACS

```
#include <kcmac.h>
#include "tp.h"
#define ERRLINE "****** Wrong TAC - Please repeat \
Input ******                                    "
void BADTACS (struct ca_area *ca, struct work *spab )
{
/* INIT-Operation                                                          */
  KDCS_SET (&spab->param, &ca->ca_head, &ca->ca_return);
  memset (&spab->std_mask.TAC,    , 8);
   KDCS_INIT (0,sizeof(struct work));
   if (KCRCC != 0 )
       { memcpy (spab->progname, "BADTACS", 8);
         errcheck (ca, spab);
       }
/* MGET-Operation                                                          */
  KDCS_MGET ( spab->std_mask.FUNCTION
             ,sizeof( FORM1 )
             ,ca->ca_return.kcrfn        );
  if (KCRCC != 0 )
         { memcpy (spab->progname, "BADTACS", 8);
           errcheck (ca, spab);
         }
/* MPUT-Operation: Replace the standard error message                      */
  memcpy (spab->std_mask.MSGTEXT, ERRLINE, 80);
  memset (spab->std_mask.TAC,    , 8);
  KDCS_MPUTNT ( &spab->std_mask
                ,sizeof( FORM1 )
                ,KDCS_SPACES
                ,"*FORM1  "
                ,KCNODF          );
  if (KCRCC != 0 )
       { memcpy (spab->progname, "BADTACS", 8);
         errcheck (ca, spab);
        }
/* PEND FI-Aufruf                                                          */
  KDCS_PENDFI();
}
```

Funktion ERRCHECK

```
 #include <kcmac.h>
 #include "tp.h"
 void errcheck ( struct ca_area * ca , struct work * spab )
 {
   struct err_line
     { char ftext[35];
       char progname[8];
       char optext[10];
       char op_code[4];
       char cctext[8];
       char cc[3];
       char cdtext[8];
       char cd[4];
     } err_msg;
/*  -------  Making connections for the KDCS_... macros   --------  */
   KDCS_SET (&spab->param, &ca->ca_head, &ca->ca_return);
 /* making entries in the errorline                                  */
   memcpy (err_msg.ftext, "******  E R R O R  in program unit ",35);
   memcpy (err_msg.progname, spab->progname, 8);
   memcpy (err_msg.optext, " Op-Code: ",10);
   memcpy (err_msg.op_code, pa.kcop, 4);
   memcpy (err_msg.cctext, " kcrccc=", 8);
   memcpy (err_msg.cc, KDCS_ERR , 3);
   memcpy (err_msg.cdtext, " kcrcdc=", 8);
   memcpy (err_msg.cd, KDCS_RTI->kcrcdc, 4);
   memset (&spab->std_mask,    , sizeof (FORM1));
   memcpy (spab->std_mask.MSGTEXT, &err_msg, 80);
 /* MPUT-Operation                                                   */
    KDCS_MPUTNE (&spab->std_mask,sizeof(FORM1),KDCS_SPACES,"*FORM1  ",KCNODF);
 /* PEND ER-Operation                                                */
   KDCS_PENDER();
}
```

Teilprogramm TPFILE mit START/SHUT-Exit und Dateizugriffen

```
  #include <stdio.h>
  #include <string.h>
  #include <stdlib.h>
  #include <kcpa.h>
  #include <kcca.h>
  #include "tp.h"
  #define M spab->std_mask
  #define TAC ca->ca_head.kccv_tac
  #define JOB TAC[0]
  #define ADR_LENGTH ( (long) sizeof( ADDRESS ) )
  #define READ     1
  #define WRITE    2
  #define UPDATE   3
  #define DELETE   4
#ifdef __SNI_HOST_BS2000
  #define FILE_NAME "link=CAPPLI"
  #define FILE_MODE "r+b,type=record,forg=key"
  #define FILE_MODE_FIRST "w+b,type=record,forg=key"
#else
```

```
  #define FILE_NAME "cappli.address"
  #define FILE_MODE "r+b"
  #define FILE_MODE_FIRST "w+b"
#endif
  #define PREVIOUS_POSITION  -ADR_LENGTH
  #define SAG_NAME       "FUJITSU TECHNOLOGY SOLUTIONS"
  #define SAG_STREET     "Musterstrasse"
  #define SAG_NUMBER     "6"
  #define SAG_PCODE      "12345"
  #define SAG_RES        "Musterstadt"
  #define SAG_PHONE      "+12 34 567-89"
  typedef enum {
                FOUND     = 1
               ,NOT_FOUND = 2
               } address_status;
  static address_status addr_fetch(struct work * );
  static ADDRESS   address;
  static FILE    * filepointer;
  static fpos_t    FilePosition;
  void TPFILE ( struct ca_area * ca , struct work *spab )
  {
    int pr_swtch;
#ifdef __SNI_HOST_BS2000
    char BS2Cmd[500];
#endif
    if (strncmp (TAC,"STARTUP ", 8) == 0)
       pr_swtch = 1;
    else

      { if (strncmp (TAC,"SHUTDOWN", 8) == 0)
            pr_swtch = 2;
        else
         { pr_swtch = 3;
           memset (M.MSGTEXT,  * , 80);
         }
      }
    switch (pr_swtch)
    { case 1:
#ifdef __SNI_HOST_BS2000
        sprintf(  BS2Cmd ,
                "SET-FILE-LINK LINK-NAME = CAPPLI ,FILE-NAME = CAPPLI.ADDRESS"\
                "             ,SUPPORT     = *DISK( SHARED-UPDATE = *YES)    ",
                ADR_LENGTH );
        system(  BS2Cmd );
#endif
        if ((filepointer = fopen (FILE_NAME, FILE_MODE)) == NULL) {
            if((filepointer = fopen( FILE_NAME , FILE_MODE_FIRST )) == NULL ) {
              perror("fopen:");
              exit(-1);
              }
            memset( &address ,     , sizeof( address ) );
            memcpy( address.NAME  , SAG_NAME   , sizeof( SAG_NAME   )-1);
            memcpy( address.STREET, SAG_STREET , sizeof( SAG_STREET )-1);
            memcpy( address.NUMBER, SAG_NUMBER , sizeof( SAG_NUMBER )-1);
            memcpy( address.POSTAL_CODE , SAG_PCODE  , sizeof( SAG_PCODE  )-1);
            memcpy( address.RESIDENCE   , SAG_RES    , sizeof( SAG_RES    )-1);
            memcpy( address.PHONE , SAG_PHONE  , sizeof( SAG_PHONE  )-1);
            fwrite(&address, ADR_LENGTH, 1, filepointer);
            fclose( filepointer );
```

```
            if ((filepointer = fopen (FILE_NAME, FILE_MODE)) == NULL) {
              perror("fopen:");
              exit(-1);
              }
            }
            break;
      case 2:
        fclose (filepointer);
        break;
      case 3:
         switch (JOB)
          { case READ:
              memcpy (M.FUNCTION, "Show address         ******", 26);
              if (addr_fetch( spab ) == NOT_FOUND)
                { memcpy (M.MSGTEXT, "Address not found      ", 22);
                  break;
                }
              else
                { memcpy (M.addr.NAME, address.NAME, ADR_LENGTH );
                  break;
                }
            case WRITE:
              memcpy (M.FUNCTION, "Enter address           ***", 26);
              if (addr_fetch( spab ) == FOUND)
                { memcpy (M.MSGTEXT, "Address already exists  ", 23);
                  break;
                }
              else
                { memcpy (address.NAME, M.addr.NAME, ADR_LENGTH );
                  fseek  (filepointer, 0, 2);
                  fwrite (&address, ADR_LENGTH, 1, filepointer);
                  memcpy (M.MSGTEXT, "Address entered          ", 24);
                  break;
                }
            case UPDATE:
              memcpy (M.FUNCTION, "Update address       ******", 26);
              if (addr_fetch( spab ) == NOT_FOUND)
                { memcpy (M.MSGTEXT, "Address not found      ", 22);
                  break;
                }
              else
                { memcpy (address.NAME, M.addr.NAME, ADR_LENGTH );
                  fsetpos(filepointer,&FilePosition );
                  fwrite (&address, ADR_LENGTH, 1, filepointer);
                  memcpy (M.MSGTEXT, "Address changed ", 16);
                  break;
                }
            case DELETE:
              memcpy (M.FUNCTION, "Delete address        *****", 26);
              if (addr_fetch( spab ) == NOT_FOUND)
                { memcpy (M.MSGTEXT, "Address not found      ", 22);
                  break;
                }
              else
                 {
#ifndef __SNI_HOST_BS2000
                  memset (&address,  * , ADR_LENGTH);
                  fsetpos(filepointer,&FilePosition );
                  fwrite (&address, ADR_LENGTH, 1, filepointer);
```

```
#else
                  fdelrec(filepointer , NULL );
                  memcpy (M.MSGTEXT, "Address deleted        ", 22);
                  break;
#endif
                }
         }
     }
      return;
  }


/*************************************************************************/
 /*                       Function addr_fetch                             */
 /*************************************************************************/
 static address_status addr_fetch( struct work *spab )
 {
   address_status filestatus = NOT_FOUND;
   memset (&address, 0X00, ADR_LENGTH);
   fseek (filepointer, 0, 0);
   while ( fgetpos( filepointer , &FilePosition ) ,
             (fread (&address, ADR_LENGTH, 1, filepointer)) != NULL)
     {
       if (address.NAME[0] !=  * )            /* Address not deleted      */
         {
           if (strncmp (address.NAME, M.addr.NAME, sizeof M.addr.NAME) == 0)
             if (strncmp (address.FIRST_NAME
                         , M.addr.FIRST_NAME
                         , sizeof M.addr.FIRST_NAME) == 0)
               { filestatus = FOUND;
                 break;
               }
         }
       memset (&address, 0X00, ADR_LENGTH);
     }
   return filestatus;
}
```

KDCDEF-Anweisungen

```
REM ***************************************************************
REM ***               D E F  -  S T A T E M E N T S               ***
REM ***                                                           ***
REM ***                      KDCFILE = CAPPLI                     ***
REM ***************************************************************
*
OPTION GEN=ALL
*
ROOT CAPPLI
*
*+-----------------------------------------------------------------+
*| MAX statements                                                  |
*+-----------------------------------------------------------------+
*
MAX KDCFILE     = kdcfile
MAX APPLINAME   = CAPPLI
MAX APPLIMODE   = S
```

```
MAX TASKS       = 3
MAX ASYNTASKS   = 1
MAX PGPOOL      = (25)
MAX CACHESIZE   = (100,30)
MAX TRACEREC    = 10000
MAX RECBUF      = (5,4096)
MAX LPUTBUF     = 10
MAX LPUTLTH     = 4096
MAX TERMWAIT    = 60
MAX KB          = 1024
MAX NB          = 2048
MAX SPAB        = 4096
MAX CLRCH       = X FF
*
*+-------------------------------------------------------------------+
*| FORMSYS statement                                                 |
*+-------------------------------------------------------------------+
*
FORMSYS ...
*
*+-------------------------------------------------------------------+
*| MESSAGE statement                                                 |
*+-------------------------------------------------------------------+
*
MESSAGE MODULE=MSGS
*
*+-------------------------------------------------------------------+
*| PROGRAM statements                                                |
*+-------------------------------------------------------------------+

REM ***    Application specific TACs            ***
*
TAC KDCMSGTC , PROGRAM=NOHACK
TAC KDCBADTC , PROGRAM=BADTACS
TAC 1        , PROGRAM=TPREAD                , LOCK = 1
TAC 2        , PROGRAM=TPUPDATE , TACCLASS=1 , LOCK = 2
TAC 3        , PROGRAM=TPUPDATE , TACCLASS=1 , LOCK = 2
TAC 4        , PROGRAM=TPUPDATE , TACCLASS=1 , LOCK = 2
*
*+-------------------------------------------------------------------+
*| TACCLASS statements                                               |
*+-------------------------------------------------------------------+
*
TACCLASS 1,TASKS=1
*
*+-------------------------------------------------------------------+
*| USER statements                                                   |
*+-------------------------------------------------------------------+
*
USER NINA  ,PASS=C SOLO  ,FORMAT=*FORM1 ,KSET=BUNDLE1                 (1)
USER URSUS               ,FORMAT=*FORM1 ,KSET=BUNDLE2                 (1)
USER ADMIN ,PASS=C ADM                  ,KSET=MASTER,PERMIT=ADMIN
*
*+-------------------------------------------------------------------+
*| TLS statements                                                    |
*+-------------------------------------------------------------------+
*
TLS    TLSB
*
```

```
*+------------------------------------------------------------------+
*| TPOOL statements                                                 |
*+------------------------------------------------------------------+
*
TPOOL LTERM=...    , NUMBER=2 , PRONAM=*ANY, PTYPE=...  , KSET = MASTER
TPOOL LTERM=...    , NUMBER=2 , PRONAM=*ANY, PTYPE=...  , KSET = MASTER
*
*+------------------------------------------------------------------+
*| PTERM / LTERM statements                                         |
*+------------------------------------------------------------------+
*
OPTION DATA=TERM
**
*+------------------------------------------------------------------+
*| KSET  statements                                                 |
*+------------------------------------------------------------------+
*

*
*
*
OPTION DATA={ PROGRAM-STATIC | PROGRAM-SHARED-OBJ | PROGRAM-BLS | PROGRAM-OLD-DLL }
*
*
*
*+------------------------------------------------------------------+
*| EXIT statements                                                  |
*+------------------------------------------------------------------+
*
EXIT PROGRAM=TPFILE ,USAGE=START
EXIT PROGRAM=TPFILE ,USAGE=SHUT
*
*+------------------------------------------------------------------+
*| TAC statements                                                   |
*+------------------------------------------------------------------+
*
REM ***         ADMINISTRATION DIALOG           ***
TAC KDCTAC    ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCLTERM  ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCPTERM  ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCSWTCH  ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCSEND   ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCAPPL   ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCUSER   ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCDIAG   ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCLOG    ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCINF    ,PROGRAM=KDCADM , ADMIN=READ
TAC KDCHELP   ,PROGRAM=KDCADM , ADMIN=READ
TAC KDCSHUT   ,PROGRAM=KDCADM , ADMIN=Y
TAC KDCTCL    ,PROGRAM=KDCADM , ADMIN=Y
*
REM ***       ADMINISTRATION  ASYNCHRON         ***
TAC KDCTACA   ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCLTRMA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCPTRMA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCSWCHA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCUSERA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCSENDA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCAPPLA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
```

```
TAC KDCDIAGA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCLOGA   ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCINFA   ,PROGRAM=KDCADM ,TYPE=A , ADMIN=READ
TAC KDCHELPA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=READ
TAC KDCSHUTA  ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
TAC KDCTCLA   ,PROGRAM=KDCADM ,TYPE=A , ADMIN=Y
*
KSET BUNDLE1 , KsEYS=(1,2)
KSET BUNDLE2 , KEYS=(1)
KSET MASTER, KEYS=MASTER
```

(1) BS2000-Format.

Input-Dateien für die Generierungsprozedur

PROGRAM-Anweisungen:

PROGRAM-STATIC (für statisches Binden)

```
PROGRAM KDCADM   ,COMP=C
PROGRAM TPREAD   ,COMP=C
PROGRAM TPUPDATE ,COMP=C
PROGRAM TPFILE   ,COMP=C
PROGRAM NOHACK   ,COMP=C
PROGRAM BADTACS  ,COMP=C
```

PROGRAM-BLS (für dynamisches Laden mit BLS)

```
* ------------------------------------------------------------------+
DEFAULT PROGRAM  COMP = C
PROGRAM KDCADM
* ------------------------------------------------------------------+
MPOOL LCPOOL , SIZE     = 10                                  -
             , SCOPE    = GROUP                               -
             , ACCESS   = READ
* ------------------------------------------------------------------+
LOAD-MODULE LLMTPS   , VERSION   = 001                         -
                     , LIB       = DYNAMIC-LOADED-LIB          -
                     , LOAD-MODE = (POOL,LCPOOL,STARTUP)
* ------------------------------------------------------------------+
DEFAULT PROGRAM COMP = C , LOAD-MODULE = LLMTPS
PROGRAM BADTACS
PROGRAM TPUPDATE
PROGRAM TPREAD
PROGRAM TPFILE
PROGRAM NOHACK
```

PTERM/LTERM-Anweisungen

```
PTERM TERM05 ,PTYPE=T9750 ,LTERM=DST01 ,PRONAM=HOST0001
LTERM DST01
PTERM TERM10 ,PTYPE=T9750 ,LTERM=DST02 ,PRONAM=HOST0001
LTERM DST02
PTERM TERM11 ,PTYPE=T9763 ,LTERM=DST03 ,PRONAM=HOST0001
LTERM DST03
PTERM TERM12 ,PTYPE=T9763 ,LTERM=DSTADMIN ,PRONAM=HOST0001
LTERM DSTADMIN
PTERM D17   ,PTYPE=T9001 ,LTERM=PRINTER1 ,PRONAM=HOST0001
LTERM PRINTER1 , USAGE = O
```

11 Ergänzungen für COBOL

Dieses Kapitel ergänzt die allgemeinen Informationen der Kapitel „Aufbau und Einsatz von UTM-Programmen" bis „Event-Funktionen" um Sprachspezifische Informationen, die Sie speziell für die Erstellung von COBOL-Teilprogrammen benötigen.

Im ersten Abschnitt erfahren Sie, wie COBOL-Teilprogramme aufgebaut sind, der zweite Abschnitt enthält Programm-Beispiele und im dritten Abschnitt sind die Datenstrukturen KCKBC und KCPAC aufgelistet.

11.1 Programmaufbau bei COBOL-Teilprogrammen

In diesem Abschnitt erfahren Sie:

- wie ein UTM-COBOL-Teilprogramm als Unterprogramm zu erstellen ist
- was Sie bei der Gestaltung der LINKAGE SECTION und der WORKING-STORAGE SECTION beachten müssen
- wie die PROCEDURE DIVISION aussehen und ein KDCS-Aufruf in COBOL programmiert werden muss
- welche Plattform-spezifischen Besonderheiten bestehen (z.B. Abhängigkeiten von spezifischen Compilern, Formatierungssystemen)

11.1.1 COBOL-Teilprogramm als Unterprogramm

UTM-Teilprogramme und Event-Exits sind Unterprogramme der UTM Main Routine.Daraus ergeben sich folgende Konsequenzen:

- Der Programmname definiert die Einsprungadresse.
- In der LINKAGE SECTION muss mindestens eine Datenstruktur definiert werden.
- Das Teilprogramm wird dynamisch mit dem PEND-Aufruf beendet; eine Ausnahme bilden die Event-Exits, die mit der Anweisung EXIT PROGRAM verlassen werden. Die Anweisung "STOP RUN" ist verboten (Ausnahme: Event-Exits START und SHUT).

Um kompatibel zu sein und mit fehlerfreien Datenstrukturen zu arbeiten, stehen Ihnen eine Reihe von COPY-Elementen zur Verfügung. Die Verwendung dieser COPY-Elemente wird in Abschnitt „Datenstrukturen für COBOL-Teilprogramme“ beschrieben.

PROGRAM-ID als Einsprungname

Im PROGRAM-ID-Paragraphen legen Sie den Einsprungnamen des Teilprogramms fest. Dieser Name ist frei wählbar. Er muss innerhalb eines Anwendungsprogramms eindeutig sein. Es dürfen keine Namenskonflikte zwischen dem Programmnamen und den Laufzeitsystemen, den Datenbanksystemen, dem Formatierungssystem, Kommunikationskomponenten und openUTM entstehen.

Bei der Namenswahl sollten Sie deshalb folgende Punkte beachten:

- Für BS2000-Systeme:
  - Alle Namen, die mit KDC, KC oder I beginnen, sind reserviert.
- Für Unix-, Linux- und Windows-Systeme:
  - Alle Namen, die mit KDC, KC , x oder ITS beginnen, sind reserviert.
  - Namen, die mit t_ beginnen, sind für CMX bzw. PCMX reserviert.
  - Namen, die mit a_, o_ oder s_ beginnen, sind für OSS reserviert.
- Der Name muss den COBOL-Konventionen entsprechen.

Den Programmnamen (Einsprungnamen) müssen Sie auch bei der Generierung der UTM-Anwendung angeben, und zwar jeweils in der KDCDEF-Anwendung PROGRAM (siehe openUTM-Handbuch „Anwendungen generieren“).

> **i** Beachten Sie die Hinweise im Handbuch des COBOL-Compilers, die sich auf die Behandlung von Programmnamen in der IDENTIFICATION DIVISION und im CALL-Aufruf beziehen.

WORKING-STORAGE SECTION

Die WORKING-STORAGE SECTION verwenden Sie vorwiegend für konstante Daten.

Um die Kompatibilität der Teilprogramme zu erreichen und ihre Lesbarkeit zu erhöhen, stehen Ihnen Konstanten mit fest vereinbarten KDCS-Namen als COPY-Elemente zur Verfügung.

Es ist sinnvoll, nur Felder mit festen Werten in die WORKING-STORAGE SECTION zu legen. Wenn Sie Ihre Bereiche mit variablen Daten ebenfalls in die WORKING-STORAGE SECTION legen möchten, können Sie hier auch den KDCS-Parameterbereich und den Nachrichtenbereich definieren. Da es aber wegen Speicherplatzeinsparung sinnvoll ist, sie im SPAB unterzubringen, werden diese Bereiche im folgenden Abschnitt beschrieben.

LINKAGE SECTION

Die LINKAGE SECTION verwenden Sie für die Parameterübergabe und als Arbeitsbereich.

Jedes Teilprogramm muss in der LINKAGE SECTION eine Datenstruktur mit Stufennummer 01 enthalten, die den KDCS-Kommunikationsbereich beschreibt.

Eine weitere Datenstruktur mit der Stufennummer 01 kann folgen. Sie beschreibt den Standard Primären Arbeitsbereich (SPAB). Im SPAB können Sie den KDCS-Parameterbereich und die Nachrichtenbereiche unterbringen.

Die Datenstrukturen von KB und KDCS-Parameterbereich sind als COPY-Elemente vorhanden (KCKBC bzw. KCPAC).

Die Nachrichtenbereiche müssen Sie selbst definieren. Für Aufrufe, die Informationen von openUTM anfordern (z. B. INFO, INIT PU) stehen jedoch spezifische Datenstrukturen in COPY-Elementen zur Verfügung. Falls Sie mit einem Formatierungssystem arbeiten, können Sie für die Strukturierung des Nachrichtenbereichs automatisch generierte Adressierungshilfen verwenden (siehe Handbuch des Formatierungssystems).

Beispiel

```
LINKAGE SECTION.
    COPY KCKBC.                        1)
    05   KB-BELIEBIG      PIC X(22).   2)
    05   KB-STARTORT      PIC X(2).    2)
    05   KB-ZIELORT       PIC X(2).    2)
    05   KB-FLGTAG        PIC X(5).    2)
    05   KB-FLGNR1        PIC X(5).    2)
    05   KB-FLGNR2        PIC X(5).    2)
    COPY KCPAC.                        3)
    03   NB.                           4)
    COPY IFORMA3.                      4)
```

1) KDCS-Kommunikationsbereich.

2) Anwender-spezifische Deklaration des KB-Programmbereichs

3) SPAB mit KDCS-Parameterbereich.

4) Nachrichtenbereich. Die COPY-Anweisung holt die Eingabe-Adressierungshilfe für das Format "FORMA3".

Erweiterung der LINKAGE SECTION

Zusätzlich zum Kommunikationsbereich und dem SPAB können Sie noch weitere Bereiche in die LINKAGE SECTION legen, die dann als gemeinsame Datenbereiche innerhalb einer UTM-Anwendung verwendet werden können.

Solche Bereiche vereinbaren Sie mit der KDCDEF-Anweisung AREA. Näheres hierzu finden Sie im openUTM-Handbuch „Anwendungen generieren“.

In COBOL-Teilprogrammen können Sie AREA-Bereiche wie folgt einsetzen:

- In der LINKAGE SECTION definieren Sie diesen Bereich mit der Stufennummer 01.
- In der PROCEDURE DIVISION geben Sie diesen Bereich bei USING mit an.

Dabei spielt auch die Reihenfolge eine Rolle, in der diese Bereiche mit der AREA-Anweisung definiert wurden. Wird der an n-ter Stelle definierte Bereich benötigt, so müssen Sie sowohl in der LINKAGE SECTION als auch in der PROCEDURE DIVISION bei USING alle Bereiche bis zu diesem angeben.

Diese Funktion gehört nicht zur DIN-Norm 66 265.

Beispiel 1

Die Bereiche BEREICH1, BEREICH2 und BEREICH3 sind in dieser Reihenfolge mit der AREA-Anweisung definiert worden. In einem Teilprogramm wird BEREICH3 benötigt. Alle Bereiche sind in der Länge 2000 definiert.

```
      .
      .
LINKAGE SECTION.
    COPY KCKBC.
      .
      .
    COPY KCPAC.
      .
      .
      .
01 BEREICH1  PIC X(2000).
01 BEREICH2  PIC X(2000).
01 BEREICH3  PIC X(2000).
      .
      .
PROCEDURE DIVISION USING KCKBC, KCSPAB, BEREICH1, BEREICH2, BEREICH3.
      .
```

Beispiel 2: Teilprogramm in COBOL (auf Unix-, Linux- und Windows-Systemen)

Im Folgenden werden zwei Areas generiert, in einer C-Source definiert (siehe "Weitere Datenbereiche (AREAs)") und an ein Teilprogramm übergeben. Definiert werden:

- die Area *area* für den direkten Zugriff, d.h. der Datenbereich wird direkt an das Teilprogramm übergeben, und

- die Area *areaind* für den indirekten Zugriff.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBAREA.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
    COPY KCKBC.
    05 PROG PIC X.
    COPY KCPAC.
    03 NB PIC X(4000).
01 AREA1 PIC X(20).
01 AREA2 PIC X(30).
PROCEDURE DIVISION USING KCKBC KCSPAB AREA1 AREA2.
    MOVE AREA1 TO BUFFER.
    MOVE AREA2 TO BUFFER1.
    PERFORM INIT-OP.
.
.
.
```

Alternative zu AREAs

Falls Teilprogramme, die AREAs verwenden, aus einer Anwendung in eine andere Anwendung übernommen werden sollen, kann die Verwendung von AREAs auf Grund ggf. unterschiedlicher Parameterleisten zu Problemen führen. Deshalb ist es empfehlenswert, an Stelle von AREAs, Datenerklärungen mit der EXTERNAL-Klausel zu verwenden. In diesem Fall entfällt die AREA-Deklaration in KDCDEF sowie die AREA-Datenerklärung in der LINKAGE SECTION und in der PROCEDURE DEVISION; stattdessen wird eine Datenerklärung mit EXTERNAL-Klausel in der WORKING-STORAGE SECTION benötigt.

Beispiel

Statt:

```
LINKAGE SECTION.
   .
   .
01 AREA1.
    02 DATA-ID  PIC X(8).
    02 DATA-EX  PIC X(4000).
```

besser:

```
WORKING-STORAGE SECTION.
01 COMMON1 IS  EXTERNAL.
    02 DATA-ID  PIC X(8).
    02 DATA-EX  PIC X(4000).
```

In diesem Beispiel ist der COMMON-Bereich COMMON1 so definiert, dass er shareable geladen werden kann. Er kann auf folgende Weise definiert werden:

Für BS2000-Systeme z.B. in Assembler

```
COMMON1  CSECT PUBLIC
COMMON1  RMODE ANY
COMMON1  AMODE ANY
*
DATA_ID  DC C'DATA-ID1'
DATA_EX  DS CL4000
         END
```

Soll der COMMON-Bereich prozesslokal geladen werden, dann kann auf das Attribut PUBLIC verzichtet werden.

Für Unix-, Linux- und Windows-Systeme in C

```
struct COMMON1 {
char DATA_ID [8] = "DATA-ID1";
char DATA_EX [4000];
}
```

11.1.2 Datenstrukturen für COBOL-Teilprogramme

Um die Datenbereiche zu strukturieren, werden mit openUTM folgende COPY-Elemente ausgeliefert, die vordefinierte Datenstrukturen enthalten.

Auf BS2000-Systemen sind die Datenstrukturen in der Bibliothek SYSLIB.UTM.070.COB enthalten.

Auf Unix-, Linux- und Windows-Systemen sind die Datenstrukturen im Verzeichnis *copy-cobol85* bzw. *netcobol* im UTM-Verzeichnis *utmpfad* enthalten.

Name	Inhalt und Bedeutung
KCAPROC	optionaler zweiter Parameterbereich für den APRO-Aufruf: Dieser Bereich dient zur Auswahl spezieller OSI TP-Funktionskombinationen und des Security-Typs.
KCATC	KDCS-Attributfunktionen (nur auf BS2000-Systemen): Bei Verwendung von +Formaten können Sie mit den symbolischen Namen für Attributfunktionen die Attributfelder der Formate verändern.
KCCFC	Nur auf BS2000-Systemen: Definiert den zweiten Parameter, den openUTM beim Event-Exit INPUT übergibt. In diesem Parameter übergibt openUTM die Inhalte der Steuerfelder von Bildschirmformaten an das Teilprogramm. Dieser zweite Parameter wird deshalb auch Steuerfeldbereich (Control Fields) genannt.
KCDADC	Datenstruktur für den DADM-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf DADM RQ über den Nachrichtenbereich legen.
KCDFC	KDCS-Bildschirmfunktionen: Mit diesen symbolischen Namen können Sie die Bildschirmausgabe beeinflussen, indem Sie den Namen der gewünschten Funktion ins Feld KCDF des KDCS-Parameterbereiches bringen.
KCINFC	Datenstruktur für den INFO-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf INFO DT/SI/PC über den Nachrichtenbereich legen.
KCINIC	definiert den zweiten Parameterbereich für den INIT-Aufruf (nur notwendig bei INIT PU) und optional für den PGWT-Aufruf (nur bei KCLI > 0). In diesen Parameterbereich liefert openUTM die angeforderten Informationen zurück.
KCINPC	Datenstruktur für den INPUT-Exit: Diese Datenstruktur enthält die Eingabe- und Ausgabeparameter des INPUT-Exits.
KCKBC	Datenstruktur für den KDCS-Kommunikationsbereich; dieser enthält: • aktuelle Daten des Vorgangs und Programms, • Rückgaben nach einem Aufruf an openUTM und • falls gewünscht den KB-Programmbereich zur Datenübergabe zwischen Programmen in einem Vorgang. Die Felder des KB-Programmbereichs müssen Sie zusätzlich definieren.

Name	Inhalt und Bedeutung
KCMSGC	Datenstruktur für die UTM-Meldungen: Diese Datenstruktur benötigen Sie, wenn Sie Meldungen in einer MSGTAC-Routine behandeln oder wenn Sie die SYSLOG-Datei mit einem eigenen Programm auswerten wollen.
KCOPC	KDCS-Operationscodes: Diese Datenstruktur enthält symbolische Namen für die KDCS-Operationen. Für Ihre KDCS-Aufrufe tragen Sie einen Namen in das Feld KCOP des KDCS-Parameterbereiches ein. Beachten Sie, dass der symbolische Name für den SIGN-Aufruf SGN lautet.
KCPAC	Datenstruktur für den KDCS-Parameterbereich: Dieser Bereich nimmt die Parameter eines KDCS-Aufrufs auf.
KCPADC	Datenstruktur für den PADM-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf PADM AI/PI über den Nachrichtenbereich legen.
KCSGSTC	Datenstruktur für den SIGN-Aufruf: Diese Datenstruktur sollten Sie beim KDCS-Aufruf SIGN ST mit KCLA > 0 über den Nachrichtenbereich legen.

Die Datenstrukturen KCOPC, KCATC, KCDFC definieren Konstanten. Deshalb kopieren Sie diese Bereiche in die WORKING-STORAGE SECTION.
Die übrigen Datenstrukturen kopieren Sie in die LINKAGE SECTION.

Die Datenstrukturen werden wie im Beispiel unten gezeigt in das Teilprogramm kopiert.

Beispiel

```
 DATA DIVISION.
*************************************************************
 WORKING-STORAGE SECTION.
     COPY KCOPC.
     COPY KCATC.  *** (nur auf BS2000-Systemen)
     COPY KCDFC.
*************************************************************
 LINKAGE SECTION.
     COPY KCKBC.
          05   KBPRG         PIC X(80).
     COPY KCPAC.
     COPY KCINFC.
          05   FILLER        PIC X(50).
     03   NB REDEFINES KCINFC.
*************************************************************
 PROCEDURE DIVISION USING KCKBC, KCSPAB.
    .
    .
```

Befehlsteil eines COBOL-Teilprogramms

Den Befehlsteil eines COBOL-Teilprogramms können Sie frei gestalten. Nur wenige Regeln der Transaktionsverarbeitung, wie sie in Kapitel „Aufbau und Einsatz von UTM-Programmen" ausführlich beschrieben sind, müssen Sie beachten:

- Die Teilprogramm sind Unterprogramm von der UTM Main Routine KDCROOT.
- Die Teilprogramme müssen reentrant-fähig sein.
- Dialog-Teilprogramme müssen den strengen Dialog einhalten.

Für Event-Exits gelten besondere Regeln, die in Kapitel "KDCS-Aufrufe in COBOL-Teilprogrammen" beschrieben werden.

Adressenübergabe

Die PROCEDURE DIVISION eines COBOL-Teilprogramms beginnt mit folgender Anweisung:

```
PROCEDURE DIVISION USING kckbc[, spab[, param1[, ... paramn]]]
```

kckbc	ist der Name des KDCS-Kommunikationsbereichs, der mit Stufennummer 01 in der LINKAGE SECTION definiert sein muss. Bei Verwendung des COPY-Elements KCKBC lautet er KCKBC.
spab	ist der Name des Standard Primären Arbeitsbereichs, der mit Stufennummer 01 in der LINKAGE SECTION definiert wurde. Bei Verwendung des COPY-Elements KCPAC lautet er KCSPAB. Wurde statt des SPAB ein Bereich der WORKING-STORAGE SECTION verwendet, so entfällt die Angabe.
$param_1$ ... $param_n$	sind die Namen weiterer Objekte, die in der LINKAGE SECTION definiert wurden, siehe auch Überschrift "Erweiterung der LINKAGE SECTION"in Abschnitt "COBOL-Teilprogramm als Unterprogramm". Diese Objekte können AREA-Bereiche sein, die als Erweiterung des SPAB dienen. Werden diese Objekte nicht verwendet, so entfällt die Angabe.

11.1.3 KDCS-Aufrufe in COBOL-Teilprogrammen

Bevor Sie im Programm eine UTM-Funktion aufrufen können, müssen alle notwendigen Parameter im KDCS-Parameterbereich stehen.

Hierzu gehören:

- der Operationscode des Aufrufs;
- zusätzliche Parameter, die durch den Operationscode bestimmt sind (siehe Kapitel „KDCS-Aufrufe“).

Bei einigen KDCS-Aufrufen müssen nicht verwendete Parameterfelder mit `LOW-VALUE` versorgt werden. Um Fehler zu vermeiden, sollten Sie immer den Befehl `MOVE LOW-VALUE TO KCPAC` absetzen, bevor die Parameterfelder versorgt werden.

Format des KDCS-Aufrufs

Wenn alle notwendigen Datenbereiche versorgt sind, kann der KDCS-Aufruf abgesetzt werden. Die Einsprungadresse für alle Operationen lautet "KDCS".

Das Format des CALL-Aufrufs ist Folgendes:

```
CALL "KDCS" USING parm1[, parm2].
```

parm1	ist der Datenname des KDCS-Parameterbereiches. Er lautet bei Verwendung des entsprechenden COPY-Elements "KCPAC". Er muss immer angegeben werden.
parm2	ist der Datenname des Speicherbereichs im Programm, in den ggf. Nachrichten oder Daten eingetragen werden sollen bzw. in dem Nachrichten oder Daten bereitgestellt wurden. In dieser Beschreibung heißt der Bereich meistens "NB" (Nachrichtenbereich). Sie können dafür aber beliebige Namen verwenden.

Die Datennamen können bei Bedarf gekennzeichnet werden.

Das Format lautet dann erweitert:

```
parm1 [{IN|OF} datenname1}...], [parm2 [{IN|OF} datenname2}...]].
```

Näheres finden Sie in der Beschreibung des COBOL-Compilers.

Beispiel

Eine mehrfach als Unterstruktur vorhandene Datenstruktur soll als Nachrichtenbereich verwendet werden.

```
.
.
   03 BUCH5.
    05 DATX             PIC X(50).
.
.
   03 BUCH8.
    05 DATX             PIC X(50).
.
.
CALL "KDCS" USING KCPAC, DATX IN BUCH5.
```

Event-Exits

Die Event-Exits INPUT, START, SHUT und VORGANG dürfen keine KDCS-Aufrufe enthalten. Sie sind als Unterprogramme zu schreiben und müssen mit der Anweisung EXIT PROGRAM beendet werden.

Bei START, SHUT und VORGANG werden die Adressen des Kommunikationsbereichs (KB) und des Standard Primären Arbeitsbereichs (SPAB) in der PROCEDURE DIVISION übergeben; dementsprechend werden die Strukturen dieser Bereiche in der LINKAGE SECTION definiert (wie bei den Teilprogrammen mit KDCS-Aufrufen). In Kapitel "Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen" finden Sie ein Beispiel für einen kombinierten START/SHUT-Exit.

Beim INPUT-Exit übergibt openUTM die Adresse des INPUT-Parameterbereichs. Für die Struktur des INPUT-Parameterbereichs steht das COPY-Element KCINPC zur Verfügung; der Name der Datenstruktur ist KCINPUTC.

Auf BS2000-Systemen kann zusätzlich die Adresse eines Steuerfeldbereichs übergeben werden. Für den Steuerfeldbereich steht das COPY-Element KCCFC zur Verfügung (Name der Struktur: KCCFILDC).

Weitere Informationen zu den Event-Exits siehe Kapitel „Event-Funktionen“.

Beispiel: INPUT-Exit

```
 DATA DIVISION.
*****************************************************************
 WORKING-STORAGE SECTION.
     .
     .
*****************************************************************
 LINKAGE SECTION.
     COPY KCINPC.
*****************************************************************
 PROCEDURE DIVISION USING KCINPUTC [,KCCFILDC].
    .
    .
```

11.1.4 Programmierung einer anwenderspezifischen Fehlerbehandlung (Unix- und Linux-Systeme)

Auf Unix- und Linux-Systemen stehen zusätzliche Funktionen zur Verfügung, die über die normale KDCS-Fehlerbehandlung hinausgehen:

- User Signal Routinen

  Dazu stehen Ihnen die beiden Funktionen KCX_REG_USER_SIGNAL_HANDLER und KCX_UN_REG_USER_SIGNAL_HANDLER für die Registrierung und De-Registrierung der User Signal Routine zur Verfügung. Die User Signal Routine wird beim Auftreten eines Signals aufgerufen und muss vom Anwender erstellt werden.
- Austausch des utmwork-Prozesses nach PEND RE mit KCX_SET_RELOAD_FLAG
- Erzeugen eines UTM-Dumps mit KCX_WRITE_DUMP

11.1.4.1 User Signal Routinen (Unix- und Linux-Systeme)

Eine User Signal Routine wird mit der Funktion KCX_REG_USER_SIGNAL_HANDLER im Teilprogramm oder im Start-Exit registriert, d.h. erst dann ist die anwenderspezifische Signalbehandlung aktiviert.

Mit KCX_UN_REG_USER_SIGNAL_HANDLER wird sie wieder de-registriert und damit deaktiviert.

Funktion KCX_REG_USER_SIGNAL_HANDLER

Die Funktion KCX_REG_USER_SIGNAL_HANDLER benötigt als Aufruf-Parameter einen Funktionspointer, der auf eine User Signal Routine zeigt. D.h. der Aufrufparameter muss vom Typ "PROCEDURE-POINTER" sein und wird automatisch per Reference, d.h. als Adresse dieses Pointers, übergeben.

Hinweis

Um diese Funktion nutzen zu können, muss Ihre COBOL Compiler Version den COBOL Datentyp "PROCEDURE-POINTER" unterstützen.
Micro Focus Visual Cobol 2.2 oder höher unterstützt z.B. diesen Datentyp.

Beispiel

```
WORKING-STORAGE section.
01 install-address-4-sig usage procedure-pointer.
PROCEDURE DIVISION USING KCKBC KCSPAB.
       set install-address-4-sig to entry "MFCBLOOP-SIGNAL".
       CALL "KCX_REG_USER_SIGNAL_HANDLER" USING install-address-4-sig.
```

Funktion KCX_UN_REG_USER_SIGNAL_HANDLER

Durch den Aufruf von KCX_UN_REG_USER_SIGNAL_HANDLER aus dem Anwendungsprogramm wird die aktuelle User Signal Routine bei openUTM wieder deregistriert, d.h. deaktiviert. Ab diesem Zeitpunkt ist wieder die Standard Fehlerbehandlung von openUTM aktiviert.

Die Funktion wird ohne Parameter aufgerufen.

Programmierung der Signal Routine

Die User Signal Routine erhält immer drei Aufrufparameter:

Param-1

Die System Signal-Nummer als binäre, 4 Byte lange Variable, d.h. Datentyp `"PIC 9(9) COMP-5"` .

Param-2

Mit Null-Byte abgeschlossene maximal 255 Bytes lange "abdruckbare" Erklärung der System Signalnummer, z.B. "Segmentation fault" bei Signal 11.

Param-3

Eine mit Null-Byte abgeschlossene, mehrzeilige und maximal 4096 Bytes lange Stack-Information.

Die User Signal Routine sollte nur "Aufräumaktionen" veranlassen und das Programm mit PEND RE beenden:

- KDCS-Aufruf RSET

- KDCS-Aufruf MPUT NE mit Fehler-Infos aus den Parametern der Anwender-Signal-Routine
- KDCS-Aufruf PEND RE mit KCRN="Folge-TAC"

Hinweise

- Die Funktion darf nicht mit "GOBACK" oder "EXIT PROGRAM" verlassen werden.
- Beim Aufruf der User Signal Routine in COBOL kann es vorkommen, dass das Micro Focus COBOL Laufzeitsystem eine Rekursion erkennt und den Aufruf ablehnt.

  Falls dies der Fall ist, muss im Anwender Teilprogramm bei der Definition der PROGRAM-ID noch der Zusatz "IS RECURSIVE" ergänzt werden, z.B.:
  PROGRAM-ID. MFCBLOOP IS RECURSIVE.

Beispiel

```
WORKING-STORAGE section.
   01 signal-number-print pic 9(9).
LINKAGE SECTION.
   01 utm-stack-string pic x(4096).
   01 utm-signal-string pic x(255).
   01 utm-signal-number pic 9(9) COMP-5.
PROCEDURE DIVISION USING KCKBC KCSPAB.
   ENTRY 'MFCBLOOP-SIGNAL' USING utm-signal-number
                                 utm-signal-string
                                 utm-stack-string.
   MOVE utm-signal-number to signal-number-print.
   DISPLAY "UTM-SIGNAL-NUMBER IS "signal-number-print.
```

Danach folgen z.B. die KDCS-Aufrufe RSET, MPUT und PEND RE.

Beispiel für an eine User Signal Routine übergebene Parameter

utm-signal-number: 11

```
utm-signal-string: "Segmentation fault"
utm-stack-string:
#6 0x00007f712cf3103f in KCSSIGNAL () from /opt/lib/utm 70a00/sys/libwork.so
#7 <signal handler called>
#8 0x00007f712e6aff30 in MFCBNULL () at MFCBNULL.c:17
#9 0x00007f712e27b418 in MFCBCALL_INIT-OP () at MFCBCALL.cbl:81
#10 0x00007f712e27b267 in MFCBCALL_INIT-OP () at MFCBCALL.cbl:59
#11 0x00007f712e27ae6a in MFCBCALL () from /home/user/filebase/utmcob.so
#12 0x00007f712e904b97 in KDCCCOB2 () from /home/user/filebase/proot.so
#13 0x00007f712cf2f7e3 in KDCHLLC () from /opt/lib/utm70a00/libwork.so
#14 0x00007f712cf8e37c in START_TEILPROGRAM () from /opt/lib/utm70a00/sys/libwork.so
#15 0x00007f712cf83658 in KDCRTMM () from /opt/lib/utm70a00/sys/libwork.so
#16 0x00007f712cf91cc3 in KDCRTSI () from /opt/lib/utm70a00/libwork.so
#17 0x00007f712cf2f36c in KDCRTST () from /opt/lib/utm70a00/libwork.so
#18 0x00007f712e90487d in kcxmnt () from /home/user/filebase/proot.so
#19 0x000000000040207c in main ()
```

11.1.4.2 utmwork-Prozess austauschen (Unix- und Linux-Systeme)

Mit der Funktion KCX_SET_RELOAD_FLAG wird der Austausch eines utmwork-Prozesses nach dem KDCS-Aufruf PEND RE veranlasst.

Es wird empfohlen, diese Funktion immer bei Cobol Laufzeitfehlern in der UTM-Anwendung aufzurufen, um die verzögerten oder verschleppten Auswirkungen dieses Fehlers zu eliminieren, d.h. die Dauerlauf-Fähigkeit der Anwendung sicherzustellen

Aufruf

```
CALL KCX_SET_RELOAD_FLAG.
```

11.1.4.3 UTM-Dump erstellen (Unix- und Linux-Systeme)

Mit der Funktion KCX_WRITE_DUMP wird das Erzeugen eines UTM-Dumps veranlasst.

Die Funktion benötigt einen Pointer auf ein 6 Byte langes Feld, das den Grund für den Dump enthält.

Beispiel für den Aufruf

```
CALL KCX_WRITE_DUMP("MYDMP1").
```

11.1.5 Plattform-spezifische Besonderheiten auf BS2000-Systemen

Hinweis zur DYNAMIC-Klausel (COBOL85)

Die DYNAMIC-Klausel sollte nicht in Lademodulen, die ausgetauscht werden sollen, verwendet werden, da der durch diese Klausel dynamisch allokierte Arbeitsspeicher bei einem Austausch nicht wieder freigegeben wird.

Beim Austausch von Lademodulen, die diese dynamische Speicherbereitstellung nutzen, kann es zu einem Speicherüberlauf kommen.

Programmieren von Teilprogrammen mit COBOL2000

Die Lebensdauer der Objekte ist auf einen Teilprogrammlauf begrenzt. Daher werden beim PEND-Aufruf alle Objekte, die innerhalb eines Teilprogrammlaufs neu angelegt wurden, vom Laufzeitsystem freigegeben. Das gilt auch bei PEND-Varianten ohne Prozess-Wechsel.

Deshalb können Objekt-Referenzen nicht über einen Teilprogrammlauf hinaus aufbewahrt werden um sie an nachfolgende Teilprogramme zu übergeben, d.h. sie können auch nicht in UTM-Speicherbereichen aufbewahrt werden.

Alle LLMs, die COBOL2000-Module mit Klassen-Definitionen enthalten, nutzen oder erben, müssen immer gemeinsam mit der modifizierten Klassen-Definition ausgetauscht werden. D.h. wenn sich eine Klassendefinition ändert, müssen alle Nutzer dieser Klasse sowie die abgeleiteten Klassen und deren Nutzer usw. gemeinsam mit der modifizierten Klassendefinition ausgetauscht werden.

Übersetzen von COBOL-Teilprogrammen

COBOL-Teilprogramme können mit dem COBOL85- oder COBOL2000-Compiler übersetzt werden, siehe „COBOL85 Benutzerhandbuch“ bzw. „COBOL2000 Benutzerhandbuch“.

Beim Übersetzen mit COBOL85 müssen Sie den COMOPT-Parameter TRUNCATE-LITERAL=NO angeben. Der COMOPT-Parameter TRUNCATE-LITERAL=NO ist bei COBOL85 ab V1.2 nicht mehr beschrieben. Der Parameter wird jedoch aus Kompatibilitätsgründen von COBOL85 noch unterstützt. Beim Übersetzen eines UTM-Teilprogramms mit COBOL85 ist die Angabe von TRUNCATE-LITERAL=NO weiterhin Pflicht.

COBOL85 meldet dann, dass der Parameter nicht dem Standard ANS85 entspricht, was aber auf das Ergebnis des Compilerlaufs keinen Einfluss hat.

Beim Übersetzen mit COBOL2000 müssen Sie den COMOPT-Parameter MARK-LAST-PARAMETER=YES angeben.

Shareable Code nutzen

Wenn Sie beabsichtigen, COBOL-Programmteile shareable zu laden, müssen Sie bereits beim Übersetzen folgende Option angeben:

```
*COMOPT GENERATE-SHARED-CODE=YES
```

Der Shareable Code muss nicht unbedingt in einem eigenen Objektmodul abgelegt werden, sondern kann zusammen mit dem nicht-shareable Teil in einem Bindelademodul (LLM) stehen, der in eine Public und eine Private Slice unterteilt ist. Hierfür ist die Compileroption

```
COMOPT GEN-LLM=YES
```

anzugeben.

Die shareable Programmteile brauchen für alle Prozesse (Tasks) der Anwendung(en) gemeinsam nur einmal geladen werden. In den task-lokalen Speicher müssen dann nur noch die nicht-shareable Teile geladen werden.

openUTM bietet verschiedene Möglichkeiten, shareable Objekte zu laden:

- als nicht-privilegiertes Subsystem,
- in einen Common Memory Pool im Benutzerspeicher (Klasse 6-Speicher).

Weitere Informationen zum Übersetzen von Shareable Code finden Sie im Benutzerhandbuch Ihres Compilers. Über das Binden und Laden von Shareable Code informiert ausführlich das openUTM-Handbuch „Einsatz von UTM-Anwendungen auf BS2000-Systemen“.

Formaterstellung mit dem IFG

Wie Sie Formate mit dem IFG erstellen können, ist ausführlich im Handbuch „IFG für FHS“ beschrieben. Wenn diese Formate für den Einsatz mit openUTM erstellt werden, so beachten Sie bitte folgende Hinweise:

- Der Formatname darf höchstens 7 Zeichen lang sein.
- Im Benutzerprofil wählen Sie die "Struktur des Datenübergabebereichs"
  - für #Formate: getrennte Attributblöcke und Feldinhalte
  - für *Formate: nicht ausgerichtet, ohne Attributfelder
  - für +Formate: nicht ausgerichtet, mit Attributfeldern
- Bei *Formaten und + Formaten vereinbaren Sie zwei Adressierungshilfen, je eine für Eingabe und Ausgabe. Definieren Sie für jede der Adressierungshilfen ein Namenspräfix. Wie Sie die Adressierungshilfen einsetzen, zeigt folgendes Beispiel:

```
LINKAGE SECTION.
  COPY KCKBC.
  05 KBPROGAREA  PIC X(100).
  COPY KCPAC.
  03   NB-EINGABE.
  COPY IFORMA-LIB.
  03   NB-AUSGABE.
  COPY OFORMA-LIB.
```

  Dabei ist FORMA der mit IFG festgelegte Formatname, I der Namenspräfix für Eingabe und O der Namenspräfix für Ausgabe. Beim Einsatz dieses Formats geben Sie beim MPUT- bzw. FPUT- oder DPUT-Aufruf den Formatnamen im Feld KCMF an als "*FORMA" (bei Adressierungshilfen ohne Attributfelder) bzw. als "+FORMA" (bei Adressierungshilfen mit Attributfeldern).

- Bitte beachten Sie bei der Definition der Adressierungshilfen, dass openUTM bei +Formaten und *Formaten zu Vorgangsbeginn beim MGET bzw. FGET den Transaktionscode aus der Nachricht entfernt (sofern dies nicht in einem INPUT-Exit explizit verhindert wird). Wenn das erste Feld im Format den Transaktionscode enthält, so müssen Sie dies bei den Adressierungshilfen für Eingabeformatierung berücksichtigen.
  Das folgende Beispiel zeigt eine Möglichkeit, wie Sie dies bei einem *Format oder +Format erreichen können:

```
LINKAGE SECTION.
   .
   .
 COPY KCPAC.
 03   NB.
                                                                    1)
   05  TACA                 PIC X(002).
   05  TAC                  PIC X(008).
   05  DATEN                PIC X(220).
 03  FILLER REDEFINES NB.
 COPY FORMA-LIB.
   .
   .
 MOVE MGET TO KCOP.
   .
 IF KCKNZVG = "F"
 THEN CALL "KDCS" USING KCPAC, DATEN        (Vorgangsanfang)
 ELSE CALL "KDCS" USING KCPAC, NB           (innerhalb des Vorgangs)
 END-IF.
```

  [1)]Bei +Formaten ist dieses Feld erforderlich, bei *Formaten muss dieses Feld entfallen.

- Bei der Einsatzvorbereitung bringen Sie die Formate in die Formateinsatzdatei (Formatbibliothek). Diesen Namen geben Sie bei den FHS-Startparametern an.

Erweiterter Zeilenmodus

Für die Arbeit im erweiterten Zeilenmodus steht das COPY-Element TIAMCTRC zur Verfügung, es wird allerdings nicht mit openUTM ausgeliefert. Dieses COPY-Element enthält die Datenstruktur LINE-MODE-CONTROL-CHARACTERS mit den symbolischen Namen für die Steuerzeichen. TIAMCTRC kann in die WORKING-STORAGE SECTION kopiert werden.

11.1.6 Plattform-spezifische Besonderheiten auf Unix- und Linux-Systemen

COBOL-Programme können Sie entweder mit den Compilern von Micro Focus oder dem Compiler NetCOBOL von Fujitsu erstellen.

Dieses Unterkapitel beschreibt folgende Compiler-spezifische Besonderheiten:

- Generierung
- Schlüsselwörter
- Umgebungsvariablen
- Übersetzen von COBOL-Teilprogrammen

Einzelheiten zum Binden einer Anwendung und zum Erzeugen von Shared Objects finden Sie im Handbuch openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“ im Unterkapitel „UTM-Prozess binden auf Unix- und Linux-Systemen".

> ¡ Alle Programme der **openUTM-Beispielanwendung** sind an die unterschiedlichen COBOL-Compiler angepasst. Je nach ausgewähltem COBOL-Compiler wird die passende Umgebung eingestellt.

Generierung

Bei der Generierung müssen Sie für diese Programme folgende KDCDEF-Anweisung angeben:

- Micro Focus COBOL:

  `PROGRAM` *`objectname`*`, COMP=MFCOBOL [,SHARED-OBJECT=`*`shared_object_name`*`]`
- NetCOBOL:

  `PROGRAM` *`objectname`*`, COMP=NETCOBOL [,SHARED-OBJECT=`*`shared_object_name`*`]`

Schlüsselwörter

Micro Focus COBOL und NetCOBOL

In den COBOL-Compilern gibt es die Schlüsselwörter OBJECT-ID, RESTRICTED und USER. Diese Schlüsselwörter kollidieren mit Namen in den COPY-Elementen der UTM-Schnittstellen. Um diese Konflikte zu vermeiden, haben Sie zwei Möglichkeiten:

- Wenn die COBOL-Teilprogramme nicht objektorientiert sind, haben Sie bei manchen Compilern die Möglichkeit, beim Übersetzen die Compiler-Schalter
  REMOVE(OBJECT-ID) , REMOVE(RESTRICTED) und REMOVE(USER) anzugeben.
- Wenn der Compiler nicht über die REMOVE-Funktionalität verfügt, bzw. wenn die objektorientierte Funktionalität erhalten bleiben soll, müssen die COPY-Anweisungen beispielsweise wie folgt modifiziert werden:

  `COPY` *`COPY-Element`* `REPLACING OBJECT-ID BY` *`NEW-OBJECT-ID`*.
  `COPY` *`COPY-Element`* `REPLACING RESTRICTED BY` *`NEW-RESTRICTED`*.
  `COPY` *`COPY-Element`* `REPLACING USER BY` *`NEW-USER`*.

  Beim Datenzugriff müssen die neuen Namen verwendet werden.

NetCOBOL

In den von openUTM ausgelieferten COBOL-Copies KCAUSERC und KCAUSD2C ist jeweils ein Strukturfeld namens PASSWORD enthalten.

Da PASSWORD im NetCOBOL-Compiler ein reserviertes Wort ist, muss beim Inkludieren dieser Dateien in der COBOL-Source eine REPLACING-Anweisung ergänzt werden, z.B.

```
COPY KCAUSERC REPLACING PASSWORD BY PASSWORD-NC.
```

Reservierte Schlüsselwörter bei Verwendung von CPIC

Bei Micro Focus COBOL ist TIMEOUT ein reserviertes Wort. Da dieses Wort aber auf Grund der CPIC-Spezifikation in dem COBOL-Copy CMCOBOL enthalten ist, muss dieser Name im Source ersetzt werden, z.B.

```
COPY CMCOBOL REPLACING TIMEOUT BY CPIC-TIMEOUT.
```

Umgebungsvariable

Micro Focus COBOL

Wenn Sie COBOL-Teilprogramme mit Micro Focus COBOL verwenden, führen Sie folgende Schritte durch:

- Rufen Sie das Skript <coboldir>/bin/cobsetenv auf. Dieses Skript setzt die notwendigen Umgebungsvariablen für den Compiler.
- Erweitern Sie die Umgebungsvariable COBCPY um $UTMPATH/copy-cobol85.
- Falls Sie Programme auf Basis von CPIC, TX bzw. XATMI unter openUTM erstellen, erweitern Sie die Umgebungsvariable COBCPY um $UTMPATH/<interface>/copy-cobol85, wobei <interface> für cpic, tx bzw. xatmi steht.
- Falls Sie Client-Programme auf Basis von UPIC-L erstellen, erweitern Sie die Umgebungsvariable COBCPY um $UTMPATH/upicl/copy-cobol85.
- Setzen Sie die Umgebungsvariable COBMODE:
  - Um 32-Bit Objekte zu erzeugen, setzen Sie sie auf 32.
  - Um 64-Bit-Objekte zu erzeugen, setzen Sie sie auf 64.

NetCOBOL

Wenn Sie NetCOBOL-Teilprogramme verwenden, führen Sie folgende Schritte durch:

- Rufen Sie das Skript <COBOLDIR>/config/cobol.sh auf. Dieses Skript setzt die notwendigen Umgebungsvariablen.
- Erweitern Sie die Umgebungsvariable COBCOPY um $UTMPATH/netcobol.
- Setzen Sie die Umgebungsvariable COB_LIBSUFFIX auf None,CPY,cpy.
- Falls Sie Programme auf Basis von CPIC, TX bzw. XATMI unter openUTM erstellen, erweitern Sie die Umgebungsvariable COB_COBCOPY um $UTMPATH/<interface>/netcobol, wobei <interface> für cpic, tx bzw. xatmi steht.
- Falls Sie Client-Programme auf Basis von UPIC-L erstellen, erweitern Sie die Umgebungsvariable COB_COBCOPY um $UTMPATH/upicl/netcobol.

Übersetzen eines COBOL-Teilprogramms

Micro Focus COBOL

Micro Focus COBOL-Quellprogramme werden mit *cob* übersetzt. Dabei geben Sie folgende Schalter an:

-c Erzeugen einer .o-Datei
-x für statisches Binden
-g damit beim Binden die Symboltabelle erhalten bleibt

NetCOBOL

NetCOBOL-Quellprogramme werden mit *cobol* erzeugt. Dabei geben Sie folgende Schalter an:

`cobol -c WC'LIST,SOURCE,XREF,MESSAGE,COPY(FULL),SRF(VAR,FIX)' P'cobolprogramm.lst' cobolprogramm.cbl` (Erzeugen einer .o-Datei)

Weitere Einzelheiten zum Erzeugen von UTM-Anwendungen mit COBOL-Programmen finden Sie im openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“.

11.1.7 Plattform-spezifische Besonderheiten auf Windows-Systemen

COBOL-Programme können Sie mit dem Compiler von Micro Focus erstellen.

Dieses Unterkapitel beschreibt folgende Compiler-spezifische Besonderheiten:

- Generierung
- Schlüsselwörter
- Umgebungsvariablen
- Übersetzen von COBOL-Teilprogrammen
- Verwendung der CPIC und XATMI COBOL-Schnittstellen

Einzelheiten zur Binden einer Anwendung finden Sie im Handbuch openUTM-Handbuch „Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen“ im Unterkapitel „UTM-Prozess binden auf Unix- und Linux-Systemen".

> ¡ Alle Programme des **openUTM Quickstart Kit** sind an den COBOL-Compiler angepasst.

Generierung

Bei der Generierung müssen Sie für diese Programme folgende KDCDEF-Anweisung angeben:
`PROGRAM` *objectname*`, COMP=MFCOBOL [,SHARED-OBJECT=`*shared_object_name*`]`

Schlüsselwörter

Im COBOL-Compiler von Micro Focus gibt es die Schlüsselwörter OBJECT-ID und RESTRICTED. Diese Schlüsselwörter kollidieren mit Namen in den COPY-Elementen der UTM-Schnittstellen. Um diese Konflikte zu vermeiden, haben Sie zwei Möglichkeiten:

- Wenn die COBOL-Teilprogramme nicht objektorientiert sind, haben Sie bei manchen Compilern die Möglichkeit, beim Übersetzen die Compiler-Schalter REMOVE(OBJECT-ID) REMOVE(RESTRICTED) und REMOVE(USER) anzugeben.
- Wenn der Compiler nicht über die REMOVE-Funktionalität verfügt, bzw. wenn die objektorientierte Funktionalität erhalten bleiben soll, müssen die COPY-Anweisungen beispielsweise wie folgt modifiziert werden:

  ```
  COPY COPY-Element REPLACING OBJECT-ID BY NEW-OBJECT-ID.
  COPY COPY-Element REPLACING RESTRICTED BY NEW-RESTRICTED.
  COPY COPY-Element REPLACING USER BY NEW-USER.
  ```

  Beim Datenzugriff müssen die neuen Namen verwendet werden.

Reservierte Schlüsselwörter bei Verwendung von CPIC

Bei Micro Focus COBOL ist TIMEOUT ein reserviertes Wort. Da dieses Wort aber auf Grund der CPIC-Spezifikation in dem COBOL-Copy CMCOBOL enthalten ist, muss dieser Name im Source ersetzt werden, z.B.

```
COPY CMCOBOL REPLACING TIMEOUT BY CPIC-TIMEOUT.
```

Umgebungsvariable

Wenn Sie Micro Focus COBOL-Teilprogramme verwenden, führen Sie folgende Schritte durch:

- Für Visual Cobol: Rufen Sie das Befehlsskript `<visualcoboldir>\base\bin\CreateEnv.bat` auf.
- Ergänzen Sie die Umgebungsvariable COBCPY um das Verzeichnis `%UTMPATH%\copy-cobol85`.
- Erweitern Sie die Umgebungsvariable INCLUDE um `<Pfad>\include`, wobei `<Pfad>` das Installationsverzeichnis des COBOL-Compilers ist (notwendig für die Übersetzung des Root Sources).
- Falls Sie Programme auf Basis von CPIC, TX bzw. XATMI unter openUTM erstellen, erweitern Sie die Umgebungsvariable COBCPY um `%UTMPATH%\<interface>\copy-cobol85`, wobei <interface> für `cpic`, `tx` bzw. `xatmi` steht.
- Falls Sie Client-Programme auf Basis von UPIC-L erstellen, erweiternSie die Umgebungsvariable COBCPY um `%UTMPATH%\upicl\copy-cobol85`.

Übersetzen eines COBOL-Teilprogramms

Programme werden übersetzt, indem Sie in einem Eingabeaufforderungsfenster das Kommando `cobol` eingeben. Soll das Teilprogramm animiert werden, muss es mit dem Kommando `cobol /ANIM` übersetzt werden.

Weitere Einzelheiten zum Übersetzen eines COBOL-Programms finden Sie in der Benutzerdokumentation des Micro Focus Compilers.

Verwendung der CPIC und XATMI COBOL-Schnittstellen

COBOL-Programme, die CPIC oder XATMI Schnittstellen nutzen, müssen wegen der verwendeten Windows-Aufrufkonventionen an den Einsatz mit einem Micro Focus Compiler angepasst werden.

- Vor der DATA DIVISION muss folgender `SPECIAL-NAMES`-Paragraph eingefügt werden, der die Aufrufkonvention `WINAPI` definiert:

  ```
  SPECIAL-NAMES.

  CALL-CONVENTION 74 is WINAPI.
  ```

- Jeder Aufruf der CPIC oder XATMI Schnittstelle muss mit dieser Konvention erfolgen, zum Beispiel:

  ```
  CALL WINAPI"CMACCP" USING CONVERSATION-ID CM_RETCODE
  ```

11.2 Programmier-Beispiele in COBOL

In diesem Abschnitt finden Sie sowohl einfache Beispiele zur Codierung eines KDCS-Aufrufs als auch ein Beispiel für eine komplette UTM-Anwendung einschließlich der KDCDEF-Generierung.

11.2.1 Beispiele zu einzelnen KDCS-Aufrufen

In diesem Abschnitt finden Sie Codierbeispiele für folgende KDCS-Aufrufe:

- MGET
- MPUT
- DPUT
- MCOM mit DPUT im Auftrags-Komplex
- APRO mit MPUT bei verteilter Verarbeitung

Da die übrigen KDCS-Aufrufe auf analoge Weise codiert werden, wird an dieser Stelle auf eine explizite Darstellung verzichtet.

Beim KDCS-Aufruf bezeichnet KCPAC die Adresse des KDCS-Parameterbereichs und NB die Adresse des Nachrichtenbereichs; es wird angenommen, dass das COPY-Element KCOPC (Konstanten für die Operationscodes) verwendet wird.

MGET-Aufruf

- Eine unformatierte Dialog-Nachricht von genau 80 Bytes Länge soll empfangen werden. Wurden weniger als 80 Zeichen eingelesen, soll eine erneute Eingabe angefordert werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MGET   TO KCOP.
MOVE 80     TO KCLA.
MOVE SPACES TO KCMF.
CALL "KDCS" USING KCPAC, NB.
 IF KCRCCC NOT = ZERO
   THEN PERFORM MGET-RETURN-CODE.  1)
 IF KCRLM NOT = KCLA
   THEN PERFORM WIEDERHOLUNG.      2)
```

1) Wurden mehr als 80 Zeichen eingelesen, wird eine Fehlerbehandlung durchgeführt.

2) In der Routine WIEDERHOLUNG wird eine Aufforderung zur Eingabe-Wiederholung an das Terminal gesendet.

- In einem laufenden Vorgang kann eine Eingabe kommen, die aus einer Kurznachricht, erzeugt mit der Funktionstaste F2, sowie aus Daten von 10 Zeichen besteht. Sie soll eine Sonderfunktion auslösen. Der Taste F2 wurde beim Generieren der Returncode 21Z zugewiesen.

```
 ...
 MOVE LOW-VALUE TO KCPAC.
 MOVE MGET TO KCOP.
 ...
 CALL "KDCS" USING KCPAC, NB.
 IF KCRCCC = "21Z"                  1)
    THEN PERFORM MGET-2.
 ...
MGET-2.                               2)
 MOVE LOW-VALUE TO KCPAC.
 MOVE MGET   TO KCOP.
 MOVE 10     TO KCLA.
 MOVE SPACES TO KCMF.
 CALL "KDCS" USING KCPAC, NB.
 IF KCRCCC NOT = ZERO
    THEN PERFORM MGET-RETURN-CODE.
```

  1) Eine Sonderfunktion wird abgefragt.

  2) Für die 10 Zeichen ist ein weiterer MGET erforderlich.

- *BS2000-Systeme:*
  Das Format "BILD15" wurde von einem Terminal angefordert. Die Länge der ungeschützten Daten beträgt 500 Zeichen in verschiedenen Formatfeldern. Dieses Format soll im Programm empfangen werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MGET       TO KCOP.
MOVE 500        TO KCLA.
MOVE "*BILD15 " TO KCMF.
CALL "KDCS" USING KCPAC, IBILD15.
IF KCRCCC = "05Z" GO TO FORMAT-FEHLER.      1)
IF KCRCCC NOT = ZERO GO TO MGET-RETURN-CODE.
...
```

  1) In der Routine ’FORMAT-FEHLER’ muss das Format nochmals ausgegeben werden, um mit dem richtigen Format weiterarbeiten zu können.

MPUT-Aufruf

- Eine unformatierte Nachricht von 80 Bytes soll an das Terminal gesendet werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MPUT   TO KCOP.
MOVE "NE"   TO KCOM.
MOVE 80     TO KCLM.
MOVE SPACES TO KCRN.
MOVE SPACES TO KCMF.
MOVE ZERO   TO KCDF.
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM MPUT-RETURN-CODE.
```

- *BS2000-Systeme:*
  Das Format "BILD15" wurde von einem Terminal angefordert. Die Länge der ungeschützten Daten beträgt 500 Zeichen in verschiedenen Formatfeldern. Dieses Format soll im Programm empfangen werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MGET       TO KCOP.
MOVE 500        TO KCLA.
MOVE "*BILD15 " TO KCMF.
CALL "KDCS" USING KCPAC, IBILD15.
IF KCRCCC = "05Z" GO TO FORMAT-FEHLER.      1)
IF KCRCCC NOT = ZERO GO TO MGET-RETURN-CODE.
...
```

  1) In der Routine ’FORMAT-FEHLER’ muss das Format nochmals ausgegeben werden, um mit dem richtigen Format weiterarbeiten zu können.

- *BS2000-Systeme:*
  Eine formatierte Nachricht von 500 Bytes soll an das Terminal geschickt werden. Der Name des *Formats ist "BILD15". Der Bildschirm soll vorher gelöscht werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MPUT      TO KCOP.
MOVE "NE"      TO KCOM.
MOVE 500       TO KCLM.
MOVE SPACES    TO KCRN.
MOVE "*BILD15" TO KCMF.
MOVE KCREPL    TO KCDF.           1)
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM MPUT-RETURN-CODE.
```

  1) REPLACE wird bei Formatwechsel standardmäßig ausgeführt. Die Ausgabe erfolgt, um Fehler wegen undefinierter Feldinhalte auszuschließen.

- *BS2000-Systeme:*
  In einem *Format "BILD10", das laut letzter Eingabe am Terminal noch vorhanden ist, sollen als Antwort alle ungeschützten Felder gelöscht werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE MPUT      TO KCOP.
MOVE "NE"      TO KCOM.
MOVE ZEROES    TO KCLM.
MOVE SPACES    TO KCRN.
MOVE "*BILD10" TO KCMF.
MOVE KCERAS    TO KCDF.
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM MPUT-RETURN-CODE.
```

DPUT-Aufruf

- Ein Asynchron-Auftrag mit einer Nachricht von 11 Zeichen soll am 11.11. (= 315. Tag im Jahr) um 11.11 Uhr an ein Teilprogramm gegeben werden (absolute Zeitangabe). Der TAC lautet "ALAAF".

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE DPUT    TO KCOP.
MOVE "NE"    TO KCOM.
MOVE 11      TO KCLM.
MOVE "ALAAF" TO KCRN.
MOVE ZERO    TO KCDF.
MOVE SPACES  TO KCMF.
MOVE "A"     TO KCMOD.
MOVE "315"   TO KCTAG.
MOVE "11"    TO KCSTD.
MOVE "11"    TO KCMIN.
MOVE "00"    TO KCSEK.
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM DPUT-RETURN-CODE.
```

- Eine Asynchron-Nachricht von 80 Zeichen soll nach 1 Stunde an das Terminal ’DSS1’ ausgegeben werden (relative Zeitangabe). Dabei soll die Bildschirmfunktion ’akustischer Alarm’ (BEL) ausgelöst werden.

```
...
MOVE LOW-VALUE TO KCPAC.
MOVE DPUT       TO KCOP.
MOVE "NE"       TO KCOM.

MOVE 80         TO KCLM.
MOVE "DSS1    " TO KCRN.
MOVE SPACES     TO KCMF.
MOVE KCALARM    TO KCDF.
MOVE "R"        TO KCMOD.
MOVE "000"      TO KCTAG.
MOVE "01"       TO KCSTD.
MOVE "00"       TO KCMIN.
MOVE "00"       TO KCSEK.
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM DPUT-RETURN-CODE.
```

Auftrags-Komplex: MCOM- und DPUT-Aufruf

Eine formatierte Asynchron-Nachricht (200 Byte) soll am selben Tag um 18.00 Uhr auf dem Drucker PRINTER2 ausgedruckt werden. Die Quittung vom Drucker wird per Programm behandelt.

Bei positiver Quittung erhält ein Asynchron-Programm mit dem TAC PRINTPOS einen Quittungsauftrag mit einer Nachricht in der Länge von 20 Byte, bei negativer Quittung wird ein Asynchron-Programm mit dem TAC PRINTNEG gestartet (ohne Nachricht). Zur negativen Quittung wird eine Benutzerinformation in der Länge von 80 Byte protokolliert. Diese kann mit DADM UI gelesen werden, sobald der Quittungsauftrag zum Hauptauftrag wird. Quittungsaufträge können nicht über eine Auftrags-ID angesprochen werden.

Der Auftrags-Komplex wird durch zwei MCOM-Aufrufe eingerahmt; dabei werden die Ziele von Druckauftrag (=Basisauftrag) und Quittungsaufträgen im Aufruf MCOM BC festgelegt; die Komplexidentifikation lautet "*PRICOMP".

```
    ...
COMPLEX-BEGIN.
    MOVE LOW-VALUE  TO KCPAC.
    MOVE MCOM       TO KCOP.
    MOVE BC         TO KCOM.
    MOVE "PRINTER2" TO KCRN.
    MOVE "PRINTPOS" TO KCPOS.
    MOVE "PRINTNEG" TO KCNEG.
    MOVE "*PRICOMP" TO KCCOMID.
    CALL "KDCS" USING KCPAC.
    IF KCRCCC NOT = ZERO
       THEN PERFORM MCOM-RETURN-CODE.
DPUT-NE.
    MOVE DPUT       TO KCOP.
    MOVE "NE"       TO KCOM.
    MOVE 200        TO KCLM.
    MOVE "*PRICOMP" TO KCRN.
    MOVE "*FORM1"   TO KCMF. *** nur auf BS2000-Systemen)
    MOVE ZERO       TO KCDF.
```

```
    MOVE "A"        TO KCMOD.
    MOVE KCTJHVG    TO KCTAG.
    MOVE "18"       TO KCSTD.
    MOVE "00"       TO KCMIN.
    MOVE "00"       TO KCSEK.
    CALL "KDCS" USING KCPAC, NB1.
    IF KCRCCC NOT = ZERO
       THEN PERFORM DPUT-RETURN-CODE.
DPUT-PLUS-T.

*****************************************************************
* Acknowledgement job in positive case
*****************************************************************
     MOVE LOW-VALUE  TO KCPAC.
     MOVE DPUT       TO KCOP.
     MOVE "+T"       TO KCOM.
     MOVE 20         TO KCLM.
     MOVE "*PRICOMP" TO KCRN.
     MOVE SPACES     TO KCMF.
     MOVE ZERO       TO KCDF.
     CALL "KDCS" USING KCPAC, NB2.
     IF KCRCCC NOT = ZERO
        THEN PERFORM DPUT-RETURN-CODE.
 DPUT-USER-INFO.
*****************************************************************
* User-Information for negative case
*****************************************************************
     MOVE LOW-VALUE  TO KCPAC.
     MOVE DPUT       TO KCOP.
     MOVE "-I"       TO KCOM.
     MOVE 80         TO KCLM.
     MOVE "*PRICOMP" TO KCRN.
     MOVE SPACES     TO KCMF.
     MOVE ZERO       TO KCDF.
     CALL "KDCS" USING KCPAC, NB3.
     IF KCRCCC NOT = ZERO
        THEN PERFORM DPUT-RETURN-CODE.
 DPUT-MINUS-T.
*****************************************************************
* Acknowledgement job in negative case
*****************************************************************
     MOVE LOW-VALUE  TO KCPAC.
     MOVE DPUT       TO KCOP.
     MOVE "-T"       TO KCOM.
     MOVE ZERO       TO KCLM.
     MOVE "*PRICOMP" TO KCRN.
     MOVE SPACES     TO KCMF.
     MOVE ZERO       TO KCDF.
     CALL "KDCS" USING KCPAC, NB4.
     IF KCRCCC NOT = ZERO
        THEN PERFORM DPUT-RETURN-CODE.
 COMPLEX-END.
     MOVE LOW-VALUE  TO KCPAC.
     MOVE MCOM       TO KCOP.
     MOVE EC         TO KCOM.
     MOVE "*PRICOMP" TO KCCOMID.
     CALL "KDCS" USING KCPAC.
     IF KCRCCC NOT = ZERO
        THEN PERFORM MCOM-RETURN-CODE.
```

```
    ...
```

Beispiel für verteilte Verarbeitung: APRO-Aufruf mit anschließendem MPUT

Vom Auftraggeber-Vorgang aus soll der Dialog-Vorgang mit dem Transaktionscode *LTAC1* der Anwendung *PARTNER1* adressiert werden (zweistufige Adressierung). Dabei soll dem Auftragnehmer-Vorgang die Vorgangs-Identifikation *>VGID1* zugeordnet werden. Anschließend wird eine MPUT-Nachricht mit Länge 100 im Zeilenmodus an die Partner-Anwendung geschickt.

```
...
MOVE LOW-VALUE  TO KCPAC.
MOVE APRO       TO KCOP.
MOVE "DM"       TO KCOM.
MOVE ZERO       TO KCLM.
MOVE "LTAC1   " TO KCRN.
MOVE "PARTNER1" TO KCPA.
MOVE ">VGID1  " TO KCPI.
CALL "KDCS" USING KCPAC.
IF KCRCCC NOT = ZERO
   THEN PERFORM APRO-RETURN-CODE.
...
MOVE LOW-VALUE  TO KCPAC.
MOVE MPUT     TO KCOP.
MOVE "NE"     TO KCOM.
MOVE 100      TO KCLM.
MOVE ">VGID1" TO KCRN.
MOVE SPACES   TO KCMF.
MOVE ZEROES   TO KCDF.
CALL "KDCS" USING KCPAC, NB.
IF KCRCCC NOT = ZERO
   THEN PERFORM MPUT-RETURN-CODE.
```

11.2.2 Beispiel für einen INPUT-Exit (BS2000-Systeme)

Der INPUT-Exit "FORINPUT" wird bei Eingaben im Formatmodus aufgerufen und reagiert auf die Eingaben wie folgt:

- Benutzerkommandos werden abgesetzt:
  - Drücken der Taste F1: KDCOUT
  - Drücken der Taste F2: KDCDISP
  - KDCOFF: erstes Zeichen der Eingabe ist "/"; wird nur außerhalb eines Vorgangs akzeptiert.
- Eine fehlende oder ungültige Eingabe wird mit einem Fehlercode in der Meldung K098 beantwortet.

Soll dem Benutzer zusätzlich die Eingabe von KDCLAST und KDCFOR erlaubt werden, muss das Programm entsprechend erweitert werden.

Dieser INPUT-Exit wird generiert mit dem Generierungstool KDCDEF in der EXIT-Anweisung mit

KDCDEF-Anweisung

```
EXIT PROGRAM=FORINPUT,USAGE=(INPUT,FORMMODE)
```

Event-Exit INPUT

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.
     FORINPUT.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 FUNC.
    05 FUNC2.
       10 COMMAND PIC X.
           88  KDCOFF   VALUE "/".
       10 REST    PIC X(7).
*
 77  KDCDISP   PIC 9(4) COMP VALUE 2.
*
 77  KDCOUT    PIC 9(4) COMP VALUE 1.
*
 77  CV-END    PIC X(2) VALUE "EC".
*
*****************************************************************
 LINKAGE SECTION.
     COPY KCINPC.
*****************************************************************
 PROCEDURE DIVISION USING KCINPUTC.
*****************************************************************
 P1-KEY-CONTROL-SECTION.
*                                      Check F-keys *
     IF KCIFKEY = KDCOUT
     THEN
        MOVE "KDCOUT"  TO KCINCMD
        MOVE "CD"      TO KCICCD
        MOVE "N"       TO KCICUT
        MOVE SPACES    TO KCIERRCD
```

```
        GO TO P99-END.
     IF KCIFKEY = KDCDISP
     THEN
        MOVE "KDCDISP" TO KCINCMD
        MOVE "CD"      TO KCICCD
        MOVE "N"       TO KCICUT
        MOVE SPACES    TO KCIERRCD
        GO TO P99-END.
 P2-CV-CONTROL.
     IF KCICVST NOT = CV-END
     THEN
        MOVE SPACES    TO KCINTAC
        MOVE "CC"      TO KCICCD
        MOVE "N"       TO KCICUT
        MOVE SPACES    TO KCIERRCD
        GO TO P99-END
     ELSE
        PERFORM P10-FUNC-CONTROL
        GO TO P99-END.
****************************************************************
 P10-FUNC-CONTROL.
****************************************************************
*                          Check the first character of input *
     MOVE KCIFCH       TO FUNC2.
     IF KDCOFF
     THEN
        MOVE "KDCOFF"  TO KCINCMD
        MOVE "CD"      TO KCICCD
        MOVE "N"       TO KCICUT
        MOVE SPACE     TO KCIERRCD
        GO TO P10-END.
     IF KCICFINF NOT = "ON"
     THEN
        MOVE SPACE     TO KCINTAC
        MOVE "ER"      TO KCICCD
        MOVE "N"       TO KCICUT
        MOVE "ER01"    TO KCIERRCD
        GO TO P10-END.
 P10-END.
     EXIT.
 P99-END.
     EXIT PROGRAM.
```

11.2.3 Beispiel für ein Asynchron-Teilprogramm MSGTAC

Das MSGTAC-Teilprogramm NOHACK zählt die Anzahl der Fehlversuche in einem TLS. Wenn openUTM ein KDCSIGN akzeptiert (d.h. Meldung K008 oder K033), so wird dieser TLS wieder gelöscht.
Falls nach drei ungültigen KDCSIGN-Versuchen der 4. KDCSIGN-Versuch wieder fehlerhaft ist, so soll das entsprechende Terminal über "asynchrone Administration" diskonnektiert werden, und zwar per FPUT-Aufruf mit KCRN="KDCPTRMA". Der Nachrichtenbereich enthält die Parameter des Administrationskommando KDCPTRMA, siehe auch openUTM-Handbuch „Anwendungen administrieren“:

```
PTERM=pterm, PRONAM=prozessor,ACT=DIS
```

Das Administrationskommando wird dann mit LPUT in der Benutzer-Protokolldatei protokolliert und der TLS gelöscht.

Die K-Meldungen werden jeweils mit FGET vom MSGTAC-Teilprogramm gelesen. Nach der "Verarbeitung" einer K-Meldung wird mit FGET sofort die nächste K-Meldung gelesen, innerhalb desselben Teilprogrammlaufs.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    MSGTAC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    COPY KCOPC.
77  ID-HACK-TLS          PIC X(8) VALUE "TLSHACK".
77  HACK-MAX             PIC 9(4) COMP VALUE 3.
01  ADM-SATZ.
  02  ADM-TXT.
    03  F                PIC X(07) VALUE "PTERM=(".
    03  F                PIC X(08).
    03  F                PIC X(09) VALUE "),PRONAM=".
    03  F                PIC X(08).
    03  F                PIC X(11) VALUE ",ACTION=DIS".
01  UTM-FEHLER-ZEILE.
    03  F                PIC X(18) VALUE "Error in prog. unit".
    03  F-MODUL          PIC X(08) VALUE "NOHACK".
    03  F                PIC X(12) VALUE "; Vorg./TAC".
    03  F-VG             PIC X(08).
    03  F                PIC X(01) VALUE "/".
    03  F-AL             PIC X(08).
    03  F                PIC X(05) VALUE " wg.".
    03  F-OP             PIC X(04).
    03  F                PIC X(07) VALUE "  (RC:".
    03  F-RC             PIC X(08).
    03  F                PIC X(01) VALUE ")".
 LINKAGE SECTION.
     COPY KCKBC.
     05  FILLER           PIC X.
     COPY KCPAC.
     COPY KCMSGC.
     03  NB.
       05  HACKER-LTERM     PIC X(8).
       05  NB-ADM.
         07  F              PIC X(07).
         07  PTRM           PIC X(08).
         07  F              PIC X(09).
         07  PRNM           PIC X(08).
```

```
         07  F              PIC X(11).
       05  TLS-HACK.
         07  HACK-ANZ       PIC 9(4) COMP.
 PROCEDURE DIVISION USING KCKBC, KCSPAB.
 MAIN SECTION .
 INIT-ANF.
     MOVE LOW-VALUE TO KCPAC
     MOVE INIT      TO KCOP
     MOVE 0         TO KCLKBPRG
     COMPUTE KCLPAB  = FUNCTION LENGTH (KCSPAB)
     CALL "KDCS" USING KCPAC.
     IF  KCRCCC  NOT =  ZERO
     THEN  GO TO  PEND-LPUT.
 FGET-ANF.
     MOVE LOW-VALUE TO KCPAC
     MOVE FGET      TO KCOP
     COMPUTE KCLA    = FUNCTION LENGTH (KCMSGC)
     MOVE SPACE     TO KCMF
     CALL "KDCS" USING KCPAC, KCMSGC
     IF  KCRCCC  NOT =  ZERO
     THEN
        IF  KCRCCC = "10Z"
        THEN
          GO TO PEND-ANF
        ELSE
          GO TO PEND-LPUT.
     IF  MSGNR = "K004"
*                                      Invalid identification *
       MOVE  LTRM OF K004  TO HACKER-LTERM
     ELSE IF  MSGNR = "K006"
*                                            Invalid password *
       MOVE  LTRM OF K006  TO HACKER-LTERM
     ELSE IF  MSGNR = "K008"
*                                            KDCSIGN accepted *
       MOVE  LTRM OF K008  TO HACKER-LTERM
     ELSE IF  MSGNR = "K031"
*                                                 Card not ok *
       MOVE  LTRM OF K031  TO HACKER-LTERM
     ELSE IF  MSGNR = "K033"
*                                     if no K008 is generated *
      MOVE  LTRM OF K033  TO HACKER-LTERM

     ELSE
       MOVE MSGNR TO KCOP
       GO TO PEND-LPUT.
     PERFORM ARBEIT
     IF  KCRCCC  NOT =  ZERO
        GO TO  PEND-LPUT.
*                                   More messages waiting ?? *
     GO TO  FGET-ANF.
 PEND-ANF.
     MOVE LOW-VALUE TO KCPAC
     MOVE PEND      TO KCOP
     MOVE "FI"      TO KCOM
     CALL "KDCS" USING KCPAC.
 PROG-ENDE.
     EXIT PROGRAM.
 PEND-LPUT.
     MOVE KCOP             TO F-OP
```

```
     MOVE KCTACVG          TO F-VG
     MOVE KCTACAL          TO F-AL
     MOVE KCRC             TO F-RC
     MOVE LOW-VALUE TO KCPAC
     MOVE LPUT      TO KCOP
     COMPUTE KCLA    = FUNCTION LENGTH (UTM-ERROR-LINE)
     CALL "KDCS" USING KCPAC, UTM-ERROR-LINE.
     MOVE LOW-VALUE TO KCPAC
     MOVE PEND      TO KCOP
     MOVE "FI"      TO KCOM
     CALL "KDCS" USING KCPAC.
 M9.
     EXIT.
/
 ARBEIT SECTION .
 A0.
     MOVE LOW-VALUE    TO KCPAC
     MOVE GTDA         TO KCOP
     MOVE 2            TO KCLA
     MOVE ID-HACK-TLS  TO KCRN
     MOVE HACKER-LTERM TO KCLT
     CALL "KDCS" USING KCPAC, TLS-HACK
     IF   KCRCCC  NOT =  ZERO
       GO TO  A9.
     IF  KCRLM = 0
     THEN
        IF  MSGNR = "K008"
              OR  = "K033"
        THEN
*                                          Ok, no TLS exists *
           NEXT SENTENCE
        ELSE
*                                                Create TLS *
           MOVE LOW-VALUE    TO KCPAC
           MOVE PTDA         TO KCOP
           MOVE 2            TO KCLA
           MOVE 1            TO HACK-NO
           MOVE ID-HACK-TLS  TO KCRN
           MOVE HACKER-LTERM TO KCLT
         CALL "KDCS" USING KCPAC, TLS-HACK
     ELSE
        IF  MSGNR = "K008"
              OR  = "K033"
        THEN
*                                            Ok; delete TLS *
           MOVE LOW-VALUE    TO KCPAC
           MOVE PTDA         TO KCOP
           MOVE 0            TO KCLA
           MOVE ID-HACK-TLS  TO KCRN
           MOVE HACKER-LTERM TO KCLT
           CALL "KDCS" USING KCPAC, TLS-HACK
        ELSE
           PERFORM CHECK-NO.
 A9.
     EXIT.
/
 PRUEF-ANZ SECTION .
 P0.
     ADD 1 TO HACK-NO
```

```
     IF  HACK-NO  NOT >  HACK-MAX
     THEN
*                                              Try it once more *
        MOVE LOW-VALUE    TO KCPAC
        MOVE PTDA         TO KCOP
        MOVE 2            TO KCLA
        MOVE ID-HACK-TLS  TO KCRN
        MOVE HACKER-LTERM TO KCLT
        CALL "KDCS" USING KCPAC, TLS-HACK
        GO TO  P9.
*                                                 Disconnect !! *
     MOVE ADM-TXT TO NB-ADM
     IF  MSGNR = "K004"
       MOVE CORR  K004 TO NB-ADM
     ELSE IF  MSGNR = "K006"
       MOVE CORR  K006 TO NB-ADM
     ELSE
       MOVE CORR  K031 TO NB-ADM.
 P-FPUT.
     MOVE LOW-VALUE  TO KCPAC
     MOVE FPUT       TO KCOP
     MOVE "NE"       TO KCOM
     MOVE "KDCPTRMA" TO KCRN
     COMPUTE KCLM     = FUNCTION LENGTH (NB-ADM)
     MOVE SPACE      TO KCMF
     MOVE ZERO       TO KCDF
     CALL "KDCS" USING KCPAC, NB-ADM
     IF  KCRCCC  NOT =  ZERO
       GO TO  P9.
 P-LPUT.
*                                           Write to user log *
     MOVE LOW-VALUE TO KCPAC
     MOVE LPUT      TO KCOP
     COMPUTE KCLA    = FUNCTION LENGTH (NB-ADM)
     CALL "KDCS" USING KCPAC, NB-ADM
     IF  KCRCCC  NOT =  ZERO
       GO TO  P9.
 P-PTDA.
*                                                    Delete TLS *
     MOVE LOW-VALUE    TO KCPAC
    MOVE PTDA         TO KCOP
    MOVE ZERO         TO KCLA
    MOVE ID-HACK-TLS  TO KCRN
    MOVE HACKER-LTERM TO KCLT
    CALL "KDCS" USING KCPAC, TLS-HACK.
P9.
    EXIT.
```

Das obige Beispiel des MSGTAC-Teilprogramm zeigt lediglich die Möglichkeiten auf, wie Meldungen geeignet ausgewertet und die Anwendung administriert werden kann.

Zur Überwachung von Sicherheitsverletzungen sollte jedoch die K094-Meldung genutzt werden (SIGNON SILENT-ALARM), da hier auch UPIC- und OSI TP-Clients eingeschlossen sind. Außerdem kann die UTM-Anwendung umfassender über die programmierte Administration (ADMI-Schnittstelle) administriert werden.

11.2.4 Beispiel für eine komplette UTM-Anwendung auf BS2000-Systemen

Beispiel Adressenverwaltung

Mit diesem Anwendungsbeispiel für eine UTM-Anwendung auf einem BS2000-System können Adressdaten verwaltet werden, die in einer Datei stehen. Die Anwendung stellt dazu die nachfolgenden Funktionen zur Verfügung, die durch Eintrag des jeweiligen TACs in das dafür vorgesehene Feld aufgerufen werden. Die Ein- und Ausgaben erfolgen in einem Format.

> ¡ Beispiele für eine UTM-Anwendung auf Unix-, Linux- oder Windows-Systemen entnehmen Sie bitte der Beispiel-Anwendung (Unix- und Linux-Systeme) und dem QuickStartKit (Windows-Systems), die zusammen mit openUTM ausgeliefert werden.

TAC	Funktion	Erklärung
1	Anzeige	gibt eine in der Datei vorhandene Adresse aus. Suchbegriff ist dabei der Name und die ersten zwei Buchstaben des Vornamens, welche in den zugehörigen Feldern anzugeben sind.
2	Neueintrag	trägt eine neue Adresse in die Datei ein. Eine Adresse mit dem gleichen Suchbegriff (s.o.) darf dort nicht schon vorhanden sein.
3	Ändern	ändert einen Adresseintrag. Die Adresse muss in der Datei schon
4	Löschen	Löscht eine in der Datei vorhandene Adresse.

Bei Fehlbedienung erscheint in der untersten Zeile des Formats eine Fehlermeldung.

Die oben genannten Ziffern sind die Transaktionscodes (TACs), die die Anwendung steuern. Dabei rufen der Transaktionscode 1 das Teilprogramm ANZEIGE und die Transaktionscodes 2, 3 und 4 das Teilprogramm AENDERN auf. Diese Teilprogramme verzweigen dann jeweils in das Teilprogramm DATEIEN. Dieses Teilprogramm wird als START- und SHUT-Exit eingesetzt und enthält die Unterprogramme, die die Ein-/Ausgaben auf die Adressdatei durchführen.

Das Teilprogramm BADTACS wird von openUTM automatisch aufgerufen, wenn ein ungültiger TAC eingegeben wird. Nach dem Aufbau der Verbindung mit der Anwendung und erfolgtem KDCSIGN wird sofort von openUTM das Format ausgegeben (Startformat). Die Arbeit mit dem Benutzer erfolgt dann im strengen Dialog, d.h. auf die Eingabe eines TACs und des Schlüssels reagiert die Anwendung mit der Ausgabe des Formats das die gesuchte Adresse enthält bzw. mit einer Erfolgs- oder einer Fehlermeldung in der untersten Zeile.

> ¡ Dieses Programm ist nur gedacht, um zu zeigen wie man mit openUTM programmiert. Die Dateizugriffe sind nicht über das UTM-Transaktionskonzept gesichert.

Die folgenden Struktogramme zeigen den Aufbau der Teilprogramme:

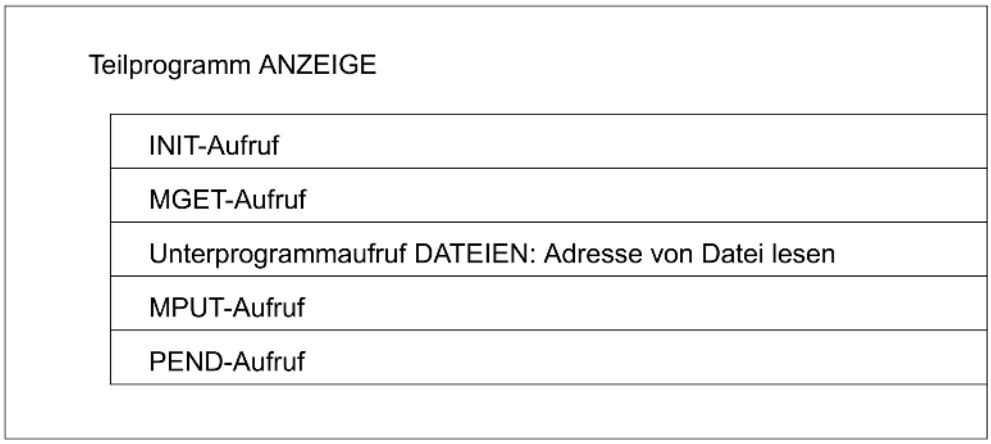


Bild: Struktogramm des Teilprogramms ANZEIGE

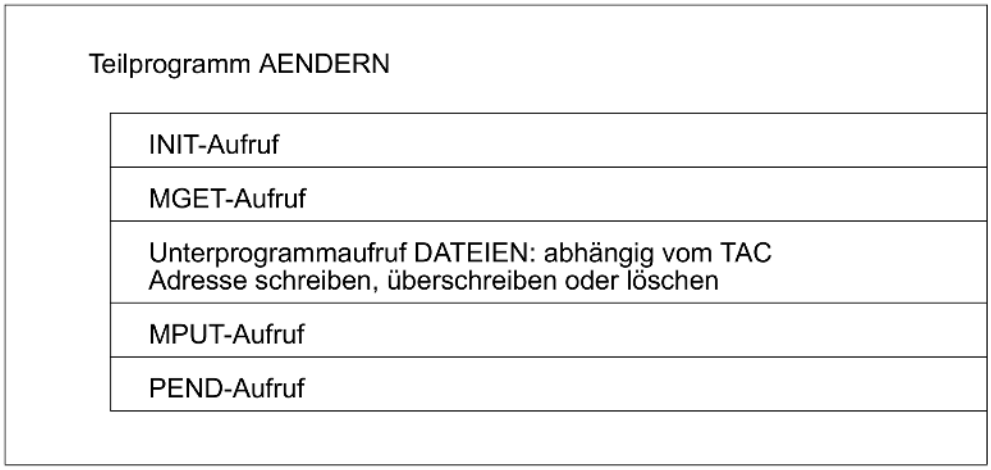
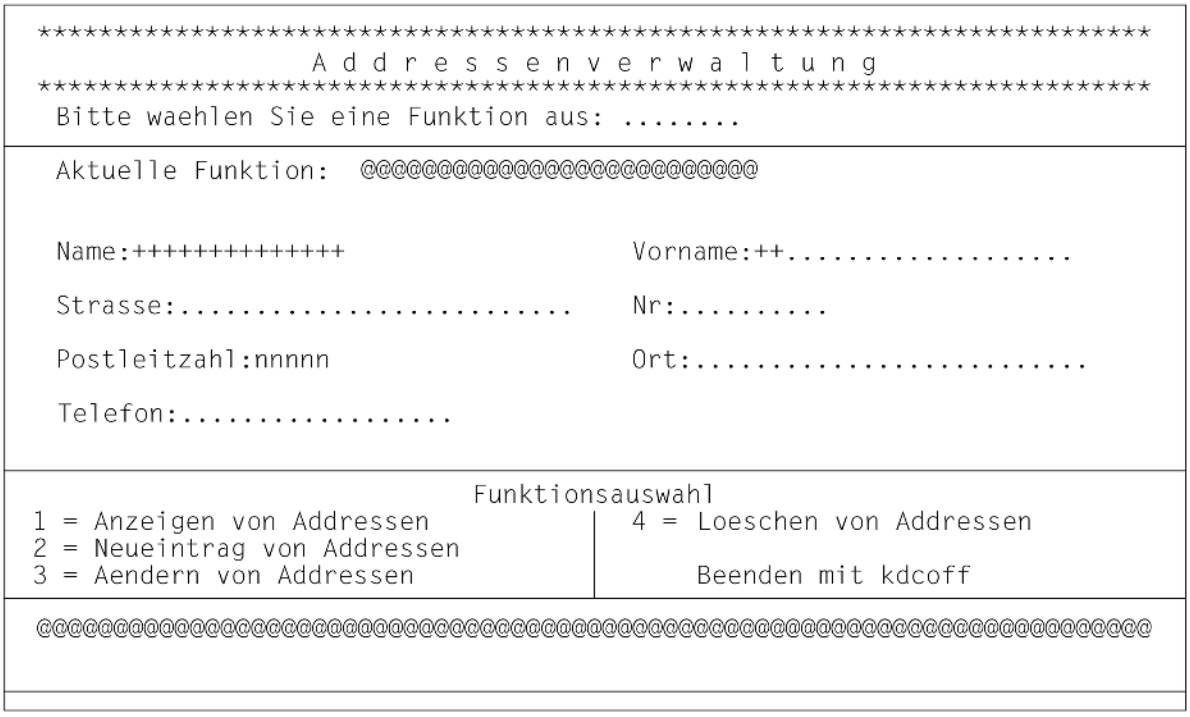


Bild: Struktogramm des Teilprogramms AENDERN

Der Vollständigkeit halber ist im Anschluss an die COBOL-Programme die Generierung dieser Anwendung aufgeführt. Die genaue Bedeutung der einzelnen Operanden und Anweisungen entnehmen Sie bitte dem openUTM-Handbuch „Anwendungen generieren“.

Das folgende Bild zeigt das Format, das bei dieser Anwendung verwendet wurde:


A d d r e s s e n v e r w a l t u n g
Bitte waehlen Sie eine Funktion aus: ....
Aktuelle Funktion: @@@@@@@@@@@@@@@@@@@@@@@@@@
Name:+++++++++++++++
Vorname:++..................
Strasse:.........................
Nr:.........
Postleitzahl:nnnnn
Ort:.........................
Telefon:.................
Funktionsauswahl
1 = Anzeigen von Addressen
2 = Neueintrag von Addressen
3 = Aendern von Addressen
4 = Loeschen von Addressen
Beenden mit kdcoff
@@@@@@@@@@@@@@@@@@@

```
            VALUE "***  F E H L E R  ***".
    05      FILLER          PIC X(14)
            VALUE "TEILPROGRAMM: ".
    05      F-TP            PIC X(08).
    05      FILLER          PIC X(17)
            VALUE " OPERATIONSCODE: ".
    05      F-OP            PIC X(04).
    05      FILLER          PIC X(13)
            VALUE " RETURNCODE: ".
    05      F-CD            PIC X(03).
 LINKAGE SECTION.
     COPY KCKBC.
     05   KBPRG                PIC X(228).
     COPY KCPAC.
     03   NB.
       05  TAC                 PIC X(008).
       05  DATEN               PIC X(220).
     03  FILLER REDEFINES NB.
     COPY FORMAO.
*************************************************************
 PROCEDURE DIVISION USING KCKBC KCSPAB.
 INIT-OPERATION-SECTION.
     MOVE SPACES   TO  NB.
     MOVE INIT     TO  KCOP.
     MOVE 0        TO  KCLKBPRG.
     MOVE 512      TO  KCLPAB.
     CALL "KDCS"       USING KCPAC.
     IF KCRCCC NOT = ZERO
        THEN MOVE INIT TO F-OP GO TO FEHLER-BEHANDLUNG.
 MGET-OPERATION.
     MOVE MGET     TO KCOP.
     MOVE 228      TO KCLA.
     MOVE "*FORMA" TO KCMF.
     CALL "KDCS"   USING KCPAC DATEN.

     IF KCRCCC NOT = ZERO
        THEN MOVE MGET TO F-OP GO TO FEHLER-BEHANDLUNG.
* AUFRUF DES TEILPROGRAMMS "DATEIEN", UM DIE LESEROUTINE   *
* AUFZURUFEN                                               *
 LESE-OPERATION.
     CALL "DATEIEN"    USING  KCKBC, KCSPAB.
MPUT-OPERATION.
     MOVE MPUT     TO  KCOP.
     MOVE "NE"     TO  KCOM.
     MOVE 228      TO  KCLM.
     MOVE SPACES   TO  KCRN.
     MOVE "*FORMA" TO  KCMF.
     CALL "KDCS" USING KCPAC NB.
     IF KCRCCC NOT = ZERO
        THEN MOVE MPUT TO F-OP GO TO FEHLER-BEHANDLUNG.
 PEND-OPERATION.
     MOVE PEND     TO  KCOP.
     MOVE "FI"     TO  KCOM.
     CALL "KDCS" USING KCPAC NB.
 PROG-ENDE.
     EXIT  PROGRAM.
 FEHLER-BEHANDLUNG.
     MOVE "ANZEIGE"   TO F-TP.
     MOVE KCRCCC      TO F-CD.
```

```
    MOVE FEHLERTEXT  TO NB.
    MOVE MPUT        TO KCOP.
    MOVE "NE"        TO KCOM.
    MOVE 80          TO KCLM.
    MOVE SPACES      TO KCRN.
    MOVE SPACES      TO KCMF.
    MOVE ZEROES      TO KCDF.
    CALL "KDCS"      USING KCPAC NB.
    MOVE PEND        TO KCOP.
    MOVE "ER"        TO KCOM.
    CALL "KDCS"      USING KCPAC.
        GO TO PROG-ENDE.
```

Teilprogramm AENDERN

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.     AENDERN.
*****************************************************************
 ENVIRONMENT DIVISION.
*****************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.
    COPY KCOPC.
 01  FEHLERTEXT.
    05     FILLER          PIC X(21)
           VALUE "***  F E H L E R  ***".
    05     FILLER          PIC X(14)
           VALUE "TEILPROGRAMM: ".
    05     F-TP            PIC X(08).
    05     FILLER          PIC X(17)
           VALUE " OPERATIONSCODE: ".
    05     F-OP            PIC X(04).
    05     FILLER          PIC X(13)
           VALUE " RETURNCODE: ".
    05     F-CD            PIC X(03).
 LINKAGE SECTION.
    COPY KCKBC.
    05   KBPRG                PIC X(228).
    COPY KCPAC.
    03   NB.
      05  TAC                 PIC X(008).
      05  DATEN               PIC X(220).
    03  FILLER REDEFINES NB.
    COPY FORMAO.
*****************************************************************
 PROCEDURE DIVISION USING KCKBC, KCSPAB.
*************************************
 INIT-OPERATION-SECTION.
    MOVE SPACES   TO  NB.
    MOVE INIT     TO  KCOP.
    MOVE 0        TO  KCLKBPRG.
    MOVE 512      TO  KCLPAB.
    CALL "KDCS"       USING KCPAC.
    IF KCRCCC NOT = ZERO
       THEN MOVE INIT TO F-OP GO TO FEHLER-BEHANDLUNG.
 MGET-OPERATION.
    MOVE MGET     TO KCOP.
```

```
     MOVE 228      TO KCLA.

     MOVE "*FORMA" TO KCMF.
     CALL "KDCS"   USING KCPAC DATEN.
     IF KCRCCC NOT = ZERO
        THEN MOVE MGET TO F-OP GO TO FEHLER-BEHANDLUNG.
* AUFRUF DES TEILPROGRAMMS "DATEIEN", UM DORT ABHAENGIG VOM TAC  *
* ZU DEN ROUTINEN SCHREIBEN, UEBERSCHREIBEN UND LOESCHEN ZU      *
* VERZWEIGEN                                                     *
 DATEI-OPERATION.
     CALL "DATEIEN" USING KCKBC, KCSPAB.
MPUT-OPERATION.
     MOVE MPUT     TO  KCOP.
     MOVE "NE"     TO  KCOM.
     MOVE 228      TO  KCLM.
     MOVE SPACES   TO  KCRN.
     MOVE "*FORMA" TO  KCMF.
     CALL "KDCS" USING KCPAC NB.
     IF KCRCCC NOT = ZERO
        THEN MOVE MPUT TO F-OP GO TO FEHLER-BEHANDLUNG.
 PEND-OPERATION.
     MOVE PEND     TO  KCOP.
     MOVE "FI"     TO  KCOM.
     CALL "KDCS" USING KCPAC NB.
 PROG-ENDE.
     EXIT  PROGRAM.
 FEHLER-BEHANDLUNG.
     MOVE "AENDERN"   TO F-TP.
     MOVE KCRCCC      TO F-CD.
     MOVE FEHLERTEXT  TO NB.
     MOVE MPUT        TO KCOP.
     MOVE "NE"        TO KCOM.
     MOVE 80          TO KCLM.
     MOVE SPACES      TO KCRN.
     MOVE SPACES      TO KCMF.
     MOVE ZEROES      TO KCDF.
     CALL "KDCS"      USING KCPAC NB.
     MOVE PEND        TO KCOP.
     MOVE "ER"        TO KCOM.
     CALL "KDCS"      USING KCPAC.
     GO TO PROG-ENDE.
```

Teilprogramm DATEIEN mit START-/SHUT-EXIT und Dateizugriffen

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    DATEIEN
 ENVIRONMENT DIVISION.
**********************
 INPUT-OUTPUT SECTION.
*---------------------
 FILE-CONTROL.
     SELECT ADRESSEN ASSIGN TO "adressen"
     ACCESS MODE IS RANDOM
     ORGANIZATION IS INDEXED
     RECORD KEY IS D-NAME
     FILE STATUS IS DATEISTATUS.
 DATA DIVISION.
```

```
****************
 FILE SECTION.
*--------------
 FD   ADRESSEN       LABEL RECORD IS STANDARD.
 01  D-ADRESSATZ.
   05  D-NAME.
     10  D-NACHNAME                  PIC X(14).
     10  D-VNS                       PIC X(02).
   05  D-VORNAME                     PIC X(18).
   05  D-STRASSE                     PIC X(26).
   05  D-NR                          PIC X(10).
   05  D-PLZ                         PIC X(04).
   05  D-ORT                         PIC X(24).
   05  D-TEL                         PIC X(16).
 WORKING-STORAGE SECTION.
*------------------------
   01 DATEIFEHLERZEILE.
   05  FILLER                        PIC X(24)
       VALUE "   *** DATEIFEHLER NR.: ".
   05  DATEISTATUS                   PIC X(02).
   05  FILLER                        PIC X(04)
       VALUE " ***".
   05  FILLER                        PIC X(50) VALUE SPACES.
 LINKAGE SECTION.

*-----------------
     COPY KCKBC.
     05   KBPRG                 PIC X(228).
     COPY KCPAC.
     03   NB.
       05  TAC                  PIC X(008).
       05  DATEN                PIC X(220).
     03  FILLER REDEFINES NB.
     COPY FORMAO.
 PROCEDURE DIVISION USING KCKBC KCSPAB.
*************************************
 STEUER SECTION.
*-----------------
 STEUER-BEGIN.
     IF KCTACVG = "STARTUP"
     THEN OPEN I-O ADRESSEN GO TO STEUER-END.
     IF KCTACVG = "SHUTDOWN"
     THEN CLOSE ADRESSEN GO TO STEUER-END.
     IF KCTACVG = "1"
     THEN GO TO LESEN-BEGIN.
     IF KCTACVG = "2"
     THEN GO TO SCHREIBEN-BEGIN.
     IF KCTACVG = "3"
     THEN GO TO UEBERSCHREIBEN-BEGIN.
     IF KCTACVG = "4"
     THEN GO TO LOESCHEN-BEGIN.
 STEUER-END.
     EXIT PROGRAM.
 LESEN SECTION.
*---------------------------------------
 LESEN-BEGIN.
*    VERSORGEN DES ISAM-SCHLUESSELS
     MOVE NAMEO                     TO D-NACHNAME.
     MOVE VNSO                      TO D-VNS.
```

```
     MOVE SPACES                    TO STRASSEO NRO ORTO TELO.
     MOVE ZEROES                    TO PLZO.
     MOVE KCTACVG                   TO TACO.
     MOVE "ANZEIGE VON ADRESSEN. "   TO FUNKTIONO.
     READ ADRESSEN RECORD
     INVALID KEY PERFORM DATEIFEHLER  GO TO LESEN-END.
     MOVE D-NACHNAME TO NAMEO.
     MOVE D-VNS      TO VNSO.
     MOVE D-VORNAME  TO VNSRESTO.
     MOVE D-STRASSE  TO STRASSEO.
     MOVE D-NR       TO NRO.
     MOVE D-PLZ      TO PLZO.
     MOVE D-ORT      TO ORTO.
     MOVE D-TEL      TO TELO.
 LESEN-END.
     EXIT PROGRAM.
 SCHREIBEN SECTION.
*-------------------------------------------
 SCHREIBEN-BEGIN.
     ENTRY   "SCHREIBEN"       USING  ADRESSATZ.
     MOVE VNSO                            TO D-VNS.
     MOVE VNSRESTO                        TO D-VORNAME.
     MOVE STRASSEO                        TO D-STRASSE.
     MOVE NRO                             TO D-NR.
     MOVE PLZO                            TO D-PLZ.
     MOVE ORTO                            TO D-ORT.
     MOVE TELO                            TO D-TEL.
     MOVE KCTACVG                         TO TACO.
     MOVE "NEUEINTRAG VON ADRESSEN. "       TO FUNKTIONO.
     MOVE " *  ADRESSE IST EINGETRAGEN  * " TO MELDTEXTO.
     WRITE D-ADRESSATZ INVALID KEY PERFORM DATEIFEHLER.
 SCHREIBEN-END.
     EXIT PROGRAM.
 UEBERSCHREIBEN SECTION.
*-------------------------------------------
 UEBERSCHREIBEN-BEGIN.
*    Satz lesen zum Sperren des Satzes
     MOVE NAMEO                     TO D-NACHNAME.
     MOVE VNSO                      TO D-VNS.
     MOVE "AENDERN VON ADRESSEN. "   TO FUNKTIONO.
     READ ADRESSEN RECORD
     INVALID KEY PERFORM DATEIFEHLER  GO TO UEBERSCHREIBEN-END.
     MOVE VNSRESTO                        TO D-VORNAME.
     MOVE STRASSEO                        TO D-STRASSE.
     MOVE NRO                             TO D-NR.
     MOVE PLZO                            TO D-PLZ.
     MOVE ORTO                            TO D-ORT.
     MOVE TELO                            TO D-TEL.
     MOVE " *  ADRESSE IST GEAENDERT  * "   TO MELDTEXTO.
     REWRITE D-ADRESSATZ INVALID KEY PERFORM DATEIFEHLER.
 UEBERSCHREIBEN-END.
     EXIT PROGRAM.
 LOESCHEN SECTION.
*-------------------------------------------
 LOESCHEN-BEGIN.
*    Satz lesen zum Sperren des Satzes
     MOVE NAMEO                   TO D-NACHNAME.
     MOVE VNSO                    TO D-VNS.
     MOVE "LOESCHEN VON ADRESSEN" TO FUNKTIONO.
```

```
     READ ADRESSEN RECORD
     INVALID KEY PERFORM DATEIFEHLER  GO TO LOESCHEN-END.
     DELETE ADRESSEN RECORD
     INVALID KEY PERFORM DATEIFEHLER  GO TO LOESCHEN-END.
     MOVE KCTACVG                   TO TACO.
     MOVE "*  ADRESSE GELOESCHT  *" TO MELDTEXTO.
 LOESCHEN-END.
     EXIT PROGRAM.
 DATEIFEHLER SECTION.
*--------------------
 DATEIFEHLER-BEGIN.
     IF DATEISTATUS = 22 THEN
     MOVE "*** ADRESSE MIT DIESEM NAMEN SCHON VORHANDEN. ***"
     TO MELDTEXTO GO TO DATEIFEHLER-END.
     IF DATEISTATUS = 23 THEN
     MOVE "*** ADRESSE MIT DIESEM NAMEN NICHT VORHANDEN. ***"
     TO MELDTEXTO GO TO DATEIFEHLER-END.
     MOVE DATEIFEHLERZEILE TO MELDTEXTO.
 DATEIFEHLER-END.
     EXIT.
```

Teilprogramm BADTACS

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. BADTACS.
*****************************************************************
 ENVIRONMENT DIVISION.
*****************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  BTEXT                      PIC X(41) VALUE
     "FALSCHER TAC - BITTE EINGABE WIEDERHOLEN.".
 77  STAR                       PIC X(6) VALUE ALL "*".
     COPY KCOPC.
 01  FEHLERTEXT.
     05     FILLER          PIC X(21)
            VALUE "***  F E H L E R  ***".
     05     FILLER          PIC X(14)
            VALUE "TEILPROGRAMM: ".
     05     F-TP            PIC X(08).
     05     FILLER          PIC X(17)
            VALUE " OPERATIONSCODE: ".
     05     F-OP            PIC X(04).
     05     FILLER          PIC X(13)
            VALUE " RETURNCODE: ".
     05     F-CD            PIC X(03).
 LINKAGE SECTION.
     COPY KCKBC.
     05     FILLER          PIC X.
     COPY KCPAC.
     03   NB.
       05  TRANSAC             PIC X(08).
       05  DATEN               PIC X(220).
     03  NB-A REDEFINES NB.
         COPY FORMAO.
     41  FEHLER REDEFINES MELDTEXTO.
       45  STAR1                 PIC X(6).
```

```
       45  BADTEXT               PIC X(41).
       45  STAR2                PIC X(6).
       45  REST                 PIC X(27).
*****************************************************************
 PROCEDURE DIVISION USING KCKBC KCSPAB.
*************************************
 INIT-OPERATION-SECTION.
     MOVE SPACES   TO  NB.
     MOVE INIT     TO  KCOP.
     MOVE 0        TO  KCLKBPRG.
     MOVE 228      TO  KCLPAB.

    CALL "KDCS"       USING KCPAC.
    IF KCRCCC NOT = ZERO
       THEN MOVE INIT TO F-OP GO TO FEHLER-BEHANDLUNG.
MGET-OPERATION.
    MOVE MGET     TO KCOP.
    MOVE 228      TO KCLA.
    MOVE "*FORMA" TO KCMF.
    CALL "KDCS"   USING KCPAC, DATEN.
    IF KCRCCC = "05Z"
       THEN MOVE SPACES TO NB-A.
       GO TO MPUT-OPERATION.
    IF KCRCCC NOT = ZERO
       THEN MOVE MGET TO F-OP GO TO FEHLER-BEHANDLUNG.
MPUT-OPERATION.
    MOVE BTEXT    TO  BADTEXT.
    MOVE STAR     TO  STAR1.
    MOVE STAR     TO  STAR2.
    MOVE SPACES   TO  REST.
    MOVE SPACES   TO  TAC.
    MOVE MPUT     TO  KCOP.
    MOVE "NE"     TO  KCOM.
    MOVE 228      TO  KCLM.
    MOVE SPACES   TO  KCRN.
    MOVE "*FORMA" TO  KCMF.
    CALL "KDCS" USING KCPAC, NB.
    IF KCRCCC NOT = ZERO
       THEN MOVE MPUT TO F-OP GO TO FEHLER-BEHANDLUNG.
PEND-OPERATION.
    MOVE PEND     TO  KCOP.
    MOVE "FI"     TO  KCOM.
    CALL "KDCS" USING KCPAC.
PROG-ENDE.
    EXIT  PROGRAM.
FEHLER-BEHANDLUNG.
        MOVE "BADTACS"   TO F-TP.
    MOVE KCRCCC      TO F-CD.
    MOVE FEHLERTEXT  TO NB.
    MOVE MPUT        TO KCOP.
    MOVE "NE"        TO KCOM.
    MOVE 80          TO KCLM.
    MOVE SPACES      TO KCRN.
    MOVE SPACES      TO KCMF.
    MOVE ZEROES      TO KCDF.
    CALL "KDCS"      USING KCPAC NB.
    MOVE PEND        TO KCOP.
    MOVE "ER"        TO KCOM.
    CALL "KDCS"      USING KCPAC.
```

```
    GO TO PROG-ENDE.
```

Generierung des Anwendungsbeispiels

```
REM ************************************************************
REM ***           D E F  -  A N W E I S U N G E N            ***
REM ***                                                      ***
REM ***                  KDCFILE = APPLI                     ***
REM ************************************************************
MAX APPLINAME=A
MAX KDCFILE=(KDCFILE.APPLI,S),TASKS=2,ASYNTASKS=0
MAX CONRTIME=5,LOGACKWAIT=60
ROOT ADR1ROOT
OPTION GEN=ALL
REM ************************************************************
REM ************        PROGRAM-ANWEISUNGEN         ************
REM ************************************************************
PROGRAM KDCADM,COMP=C
PROGRAM TPREAD,COMP=COB1
PROGRAM TPUPDATE,COMP=COB1
PROGRAM TPFILE,COMP=COB1
PROGRAM BADTACS,COMP=COB1
REM ************************************************************
REM ************         EXIT-ANWEISUNGEN           ************
REM ************************************************************
EXIT PROGRAM=TPFILE,USAGE=START
EXIT PROGRAM=TPFILE,USAGE=SHUT
REM ************************************************************
REM ************          TAC-ANWEISUNGEN           ************
REM ************************************************************
DEFAULT TAC ADMIN=Y,PROGRAM=KDCADM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
DEFAULT TAC TYPE=A,ADMIN=Y,PROGRAM=KDCADM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA

TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
```

```
TAC KDCTCLA
DEFAULT TAC TYPE=D,PROGRAM=(STD)
TAC KDCBADTC,PROGRAM=BADTACS
TAC 1,LOCK=1,PROGRAM=TPREAD
TAC 2,LOCK=2,PROGRAM=TPUPDATE
TAC 3,LOCK=2,PROGRAM=TPUPDATE
TAC 4,LOCK=2,PROGRAM=TPUPDATEREM
*************************************************************
REM ************         USER-ANWEISUNGEN           ************
REM *************************************************************
USER  GUENTER,PASS=C'AUFGEHTS',KSET=BUND1,PERMIT=ADMIN,FORMAT=*FORMA
USER  BESSY,PASS=C'HH',KSET=BUND2,STATUS=ON,FORMAT=*FORMA
USER  HAPPI,KSET=BUND3,STATUS=ON,FORMAT=*FORMA
REM *************************************************************
REM ************         FORMSYS-ANWEISUNG          ************
REM *************************************************************
FORMSYS TYPE=FHS
REM *************************************************************
REM ************      PTERM/LTERM-ANWEISUNGEN       ************
REM *************************************************************
DEFAULT PTERM PRONAM=DSR01,PTYPE=T9750
PTERM DSS01,LTERM=UTMDST1
PTERM DSS02,LTERM=UTMDST2
PTERM DSS03,LTERM=UTMDST3
DEFAULT PTERM PRONAM=DSR01,PTYPE=T9022,USAGE=O
PTERM G01,LTERM=DRUCKER,CONNECT=A
LTERM UTMDST1,KSET=BUND1
LTERM UTMDST2,LOCK=4,KSET=BUND1
LTERM UTMDST3,LOCK=5,KSET=BUND1
LTERM DRUCKER,USAGE=O
REM *************************************************************
REM ************         KSET-ANWEISUNGEN           ************
REM *************************************************************
KSET  BUND1,KEYS=(1,2,3,4,5)
KSET  BUND2,KEYS=(1,2,4)
KSET  BUND3,KEYS=(1)
REM *************************************************************
REM ************          TLS-ANWEISUNGEN           ************
REM *************************************************************
TLS    TLSA
TLS    TLSB
END
```

12 Anhang

- Übersicht über alle KDCS-Aufrufe
- Unterschiedliche Feldnamen für C/C++ und COBOL
- ASCII - EBCDIC Code-Konvertierung
  - BS2000-Systeme
  - Unix-, Linux- und Windows-Systeme
    - Code-Konvertierungstabellen auf Unix- und Linux-Systemen modifizieren
    - Code-Konvertierungstabellen auf Windows-Systemen modifizieren

12.1 Übersicht über alle KDCS-Aufrufe

Überblick über die Einträge im KDCS-Parameterbereich und Nachrichtenbereich (NB) bei den KDCS-Aufrufen. Nicht aufgeführte Felder sind mit dem Wert binär Null zu versorgen.

In der folgenden Tabelle sind die Feldnamen in C/C++ nur aufgeführt, wenn sie sich von den Feldnamen in COBOL durch mehr als nur die Groß-/Kleinschreibung unterscheiden.

In der folgenden Tabelle bedeuten:

0 binär null
B Leerzeichen (Blanks)
X sonstige Angaben
* Rückgaben

KDCS-Parameterbereich								**NB / 2. Parameter-Bereich**
KCOP	**KCOM**	**KCLA KCLKBPRG /kclcapa**	**KCLM KCLPAB /kclspa**	**KCRN**	**KCMCOM**			
					KCMF /kcfn KCLT KCUS KCPA KCGTM	**KCDF KCLI KCQRC**	**KCAPRO KCDPUT KCDGET KCQCRE KCEVENT KCPADM KCSGCL /kc_sgcl KCNORPLY**	
APRO	AM DM		X X	X X	X X		X X	[X] [X]
CTRL	PR PE AB	0 0 0	0 0 0	X X X	B B B		X/0 X/0 0	derzeit nicht verwendet; muss übergeben werden
DADM	RQ UI CS DL DA MV MA	X X 0 0 0 0 0	0 0 0 0 0 0 0	X X X X B X B	X 0 0 X X X X	0 0 0 0 0 0 0	X[1] X X X[1] X[1] X X	* * * * * * *

KDCS-Parameterbereich								NB / 2. Parameter-Bereich
KCOP	**KCOM**	**KCLA** **KCLKBPRG /kclcapa**	**KCLM** **KCLPAB /kclspa**	**KCRN**	**KCMCOM**			
					KCMF /kcfn KCLT KCUS KCPA KCGTM	**KCDF KCLI KCQRC**	**KCAPRO KCDPUT KCDGET KCQCRE KCEVENT KCPADM KCSGCL /kc_sgcl KCNORPLY**	
DGET	FT	X	0	X	B	0	X	*
	NT	X	0	X	B	0	X	*
	BF	X	0	X	X	0	X	*
	BN	X	0	X	X	0	X	*
	PF	X	0	X	X	0	X	*
	PN	X	0	X	X	0	X	*
DPUT	NT		X	X	X	X	X	X
	NE		X	X	X	X	X	X
	NI	0	X	X	B	0	X	X
	QT	0	X	X	X	0	X[1]	X
	QE	0	X	X	X	0	X[1]	X
	QI	0	X	X	B	0	X	X
	+T	0	X	X	B	0	0	X
	-T	0	X	X	B	0	0	X
	+I	0	X	X	B	0	0	X
	-I	0	X	X	B	0	0	X
	RP[2]	0	X	X	B	0	X	X
FGET		X			X			*
FPUT	NT		X	X	X	X		X
	NE		X	X	X	X		X
	RP[2]		X	X	B	0		X
GTDA		X		X	X			*
INFO	CD	X			X			*
	DT	X						*
	LO	X						*
	PC	X						*
	SI	X						*
	CK							X

KDCS-Parameterbereich								NB / 2. Parameter-Bereich
KCOP	**KCOM**	**KCLA** **KCLKBPRG /kclcapa**	**KCLM** **KCLPAB /kclspa**	**KCRN**	**KCMCOM**			
					KCMF /kcfn KCLT KCUS KCPA KCGTM	**KCDF KCLI KCQRC**	**KCAPRO KCDPUT KCDGET KCQCRE KCEVENT KCPADM KCSGCL /kc_sgcl KCNORPLY**	
INIT		X	X					
	PU	X	X	0	0	X	0	*
	MD	X	0	0	0	0	0	
LPUT		X						X
MCOM	BC	0	0	X	X			
	EC	0	0	0	X			
MGET		X			X			*
	NT	X		X	X			*
MPUT	NT		X	X	X	X		X
	NE		X	X	X	X		X
	PM	0	X	B	X	X	0	X
	RM	0	X	X	0	X	0	X
	EM	0	0	X	B	0	0	X
	ES	0	X	0	X	0	0	X
	HM	0	0	X	B	0	0	X
PADM	OK	0	0	X	X	0	0	*
	PR	0	0	X	X	0	0	*
	AT	0	0	X	X	0	0	*
	AC	0	0	X	X	0	0	*
	CA	0	0	X	X	0	X	*
	CS	0	0	X	X	0	X	*
	AI	X	0	X	X	0	0	*
	PI	X	0	X	X	0	0	*

KDCS-Parameterbereich								NB / 2. Parameter-Bereich
KCOP	**KCOM**	**KCLA** **KCLKBPRG /kclcapa**	**KCLM** **KCLPAB /kclspa**	**KCRN**	**KCMCOM**			
					KCMF /kcfn KCLT KCUS KCPA KCGTM	**KCDF KCLI KCQRC**	**KCAPRO KCDPUT KCDGET KCQCRE KCEVENT KCPADM KCSGCL /kc_sgcl KCNORPLY**	
PEND	PA PR PS KP RE SP FC RS FR FI ER	 0 0 0 0	 0 0 0 0	X X X X X X X B B	 0 0 0 0	 0 0 0 0	 0 0 0 0	
PGWT	KP PR CM RB	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	X X X X	0 0 0 0	* * * *
PTDA		X		X	X			X
QCRE	NN WN	X X	0 0	B X	B B	0 0	X X	
QREL	RL	0	0	X	B	0	0	
RSET								
SGET	KP RL GB US	X X X X	 0	X X X X	 X	 0	 0	* * * *

KDCS-Parameterbereich								NB / 2. Parameter-Bereich
KCOP	**KCOM**	**KCLA** **KCLKBPRG /kclcapa**	**KCLM** **KCLPAB /kclspa**	**KCRN**	**KCMCOM**			
					KCMF /kcfn KCLT KCUS KCPA KCGTM	**KCDF KCLI KCQRC**	**KCAPRO KCDPUT KCDGET KCQCRE KCEVENT KCPADM KCSGCL /kc_sgcl KCNORPLY**	
SIGN	ST ON CP CK OF OB CL[2]	X X X X 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 X 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 X	X X X X
SPUT	DL MS ES GB US	X X X X X	0	X X X X X	X	0	0	X X X X X
SREL	LB GB			X X				
UNLK	GB DA US	0	0	X X X	X X	0	0	

[1] Steht in KCLT der Name einer USER- oder Temporären Queue, muss im Feld KCQTYP der Wert U bzw. Q angegeben werden

[2] nur auf BS2000-Systemen

Übersicht über die Rückgaben im KDCS-Kommunikationsbereich bei den KDCS-Aufrufen.

In der folgenden Tabelle bedeutet ein Stern (*) immer einen Rückgabewert

Aufruf		**KDCS-Kommunikationsbereich**							
KCOP	**KCOM**	**KB-Kopf**	**KB-Rückgabebereich**						**KB-Programm-bereich**
			KCRDF	**KCRLM**	**KCRINFCC KCRMGT KCRST KCRSIGN**	**KCRCCC KCRCDC**	**KCRMF /kcrfn**	**KCRPI KCRUS KCRWVG KCRQN KCRQRC KCRGTM KCRDPID KCRRC**	
APRO	AM DM					* *			
CTRL	PR PE AB					* * *			
DADM	RQ UI CS DL DA MV MA			* *		* * * * * * *	*		
DGET	FT NT BF BN PF PN			* * * * * *		* * * * * *	* *	* * * * * *	

Aufruf		KDCS-Kommunikationsbereich							
KCOP	KCOM	KB-Kopf	KB-Rückgabebereich						KB-Programm-bereich
			KCRDF	KCRLM	KCRINFCC KCRMGT KCRST KCRSIGN	KCRCCC KCRCDC	KCRMF /kcrfn	KCRPI KCRUS KCRWVG KCRQN KCRQRC KCRGTM KCRDPID KCRRC	
DPUT	NT NE NI QT QE QI +T -T +I -I RP[1]					* * * * * * * * * * *			
FGET				*		*	*		
FPUT	NT NE RP[1]					* * *			
GTDA				*		*			
INFO	CD DT LO PC SI			* * * * *		* * * * *			
	CK			*	*	*			
INIT	 PU MD	* *		*		* * *	* *	* *	* *
LPUT						*			
MCOM	BC EC					* *			

Aufruf		KDCS-Kommunikationsbereich							
KCOP	KCOM	KB-Kopf	KB-Rückgabebereich						KB-Programm-bereich
			KCRDF	KCRLM	KCRINFCC KCRMGT KCRST KCRSIGN	KCRCCC KCRCDC	KCRMF /kcrfn	KCRPI KCRUS KCRWVG KCRQN KCRQRC KCRGTM KCRDPID KCRRC	
MGET	NT		* *	* *	* *	* *	* *	* *	
MPUT	NT NE PM RM EM ES HM		*			* * * * * * *			
PADM	OK PR AT AC CA CS AI PI			* *		* * * * * * * *	* *		
PEND	PA PR PS KP RE SP FI FC RS ER FR					* * * * * * * * * * *			
PGWT[2]	KP PR CM RB			* * * *		* * * *	* *	* *	

Aufruf		KDCS-Kommunikationsbereich							
KCOP	**KCOM**	**KB-Kopf**	**KB-Rückgabebereich**						**KB-Programm-bereich**
			KCRDF	**KCRLM**	**KCRINFCC KCRMGT KCRST KCRSIGN**	**KCRCCC KCRCDC**	**KCRMF /kcrfn**	**KCRPI KCRUS KCRWVG KCRQN KCRQRC KCRGTM KCRDPID KCRRC**	
PTDA						*			
QCRE	NN WN					* *		*	
QREL	RL					*			
RSET						*			*
SGET	KP RL GB US			* * * *		* * * *			
SIGN	ST			*	*	*	*	*	
	ON CP CK OF OB CL[1]				* *	* * * * * *			
SPUT	DL MS ES GB US					* * * * *			
SREL	LB GB					* *			
UNLK	GB DA US					* * *			

[1] nur auf BS2000-Systemen

[2] KCRLM wird nur dann versorgt, wenn KCLI>0 angegeben wurde.

12.2 Unterschiedliche Feldnamen für C/C++ und COBOL

Im Handbuch werden für den Kommunikationsbereich und den KDCS-Parameterbereich die Feldnamen für COBOL benutzt, die immer in Großbuchstaben geschrieben werden. Für die Feldnamen in C/C++ werden grundsätzlich Kleinbuchstaben verwendet.

Da die Feldnamen für C/C++ von den englischen Begriffen abgeleitet wurden, ergeben sich neben der Groß /Kleinschreibung bei den Feldern von KB und KDCS-Parameterbereich weitere Unterschiede.

In diesem Handbuch ist überall dort, wo sich über Groß/Kleinschreibung hinausgehende Unterschiede ergeben, der Feldname für C/C++ hinter dem COBOL-Feldnamen angegeben (durch Schrägstrich getrennt), z.B.: "KCTAG /kcday".

In den folgenden Tabellen sind alle COBOL-Feldnamen, die sich durch mehr als durch die Groß/Kleinschreibung vom C/C++-Feldnamen unterscheiden, grau unterlegt.

Beachten Sie ferner, dass

- in den Datenstrukturen *kcdad.h* und *kcpad.h* die Felder für die Zeitangaben in C/C++ **nicht** zu einer Gruppe zusammengefasst sind.
- in der Datenstruktur *kcini.h* die Feldnamen anders aufgebaut sind als in der entsprechenden COBOL-Datenstruktur KCINIC (siehe Tabelle auf "INIT Initialisieren eines Teilprogramms"). Diese Datenstrukturen dienen zur Strukturierung des Nachrichtenbereichs für den Aufruf INIT mit Modifikation PU und den Aufruf PGWT mit KCLI>0.
- auf BS2000-Systgemen die Bildschirmfunktion für das Lesen vom Ausweisleser in C/C++ den symbolischen Namen KCCARDRD besitzt.

Feldnamen im KB-Kopfbereich KCKBKOPF (ca_hdr)

COBOL-Name	C/C++-Name	Bedeutung
KCBENID	kcuserid	user-identification
KCTACVG	kccv_tac	service: name of the transaction code
KCTAGVG	kccv_day	start of service: day
KCMONVG	kccv_month	start of service: month
KCJHRVG	kccv_year	start of service: year
KCTJHVG	kccv_doy	start of service: day of the year
KCSTDVG	kccv_hour	start time of service: hour
KCMINVG	kccv_minute	start time of service: minute
KCSEKVG	kccv_second	start time of service: second
KCKNZVG	kccv_status	service: status information
KCTACAL	kcpr_tac	program run: name of the transaction code

COBOL-Name	C/C++-Name	Bedeutung
KCSTDAL	kcpr_hour	start time of program run: hour
KCMINAL	kcpr_minute	start time of program run: minute
KCSEKAL	kcpr_second	start time of program run: second
KCAUSWEIS	kccard	status of card reader
KCTAIND	kctaind	transaction indicator
KCLOGTER	kclogter	logical terminalname
KCTERMN	kctermn	terminal mnemonic
KCLKBPB	kclpa	length of program area
KCHSTA	kchsta	stack level
KCDSTA	kcdsta	change in stack level
KCPRIND	kcprind	program indicator
KCOF1	kcof1	OSI TP functional unit
KCCP	kccp	client protocol
KCTARB	kctarb	transaction rollback indicator
KCYEARVG	kccv_year4	start of service: day of the year, 4-digit

Feldnamen im KB-Rückgabebereich KCRFELD (ca_rti)

COBOL-Name	C/C++-Name	Bedeutung
KCRDF	kcrdf	device feature
KCRLM	kcrlm	input message length
KCRINFCC	kcrinfcc	return informations of INFO CK
KCVGST	kcpcv_state	service state of partner
KCTAST	kcpta_state	transaction state of partner
KCRMGT	kcrmgt	type of message
KCRSIGN	kcrsign	status of SIGN ON (complete code)
KCRSIGN1	kcrsign1	primary code of SIGN ON
KCRSIGN2[1]	—	secondary code of SIGN ON
KCRCCC	kcrccc	compatible returncode
KCRCKZ	kcrcid	identifier of DC-System
KCRCDC	kcrcdc	returncode of DC-System
KCRMF	kcrfn	format name
KCRPI	kcrpi	service identification
KCRQN	kcrqn	return queue name
KCRWVG	kcrwvg	return number waiting vg
KCRUS	kcrus	return user (SIGN ST, DGET FT)
KCRQRC	kcrqrc	queue specific redelivery counter
KCRGTM	kcrgtm	creation time of DGET message
KCRDPID	kcrdpid	DPUT ID of DGET message
KCRRC	kcrrc[2]	redelivery counter of DGET message

[1] Das Feld KCRSIGN2 ist in der C/C++-Datenstruktur *ca_rti* nicht definiert; der sekundäre Code des SIGN-Aufrufs ist im zweiten und dritten Byte von *kcrinfcc* zu finden.
[2] Es gibt kein Feld in der C Datenstruktur vom passenden Typ, das direkt verwendbar wäre. Es muss das Feld KCRRC aus kcmac.h vom Typ unsigned short verwendet werden.

Feldnamen im KDCS-Parameterbereich KCPAC (kc_pa)

COBOL-Name	C/C++-Name	Bedeutung
KCOP	kcop	operationcode
KCOM	kcom	operation modification
KCLA	kcla	length of data area
KCLKBPRG	kclcapa	length of ca-program area
KCLM	kclm	length of message (part)
KCWTIME	kcwtime	waiting time for DGET messages
KCLPAB	kclspa	length of standard-primary area
KCRN	kcrn	reference name
KCMF	kcfn	format name
KCLT	kclt	logical terminal name
KCUS	kcus	name of user
KCPA	kcpa	partner application name
KCOF	kcof	OSI functions
KCDF	kcdf	device feature
KCLI	kcli	length of init area
EXTENT	kcext	extentions for DPUT, APRO and PADM
KCDPUT	kcdput	extention for DPUT function
KCMOD	kcmod	DPUT: modifier
KCTAG	kcday	DPUT: days
KCSTD	kchour	DPUT: hours
KCMIN	kcmin	DPUT: minutes
KCSEK	kcsec	DPUT: seconds
KCQTYP	kcqtyp	queue type
KCQMODE	kcqmode	queue mode
KCAPRO	kcapro	extention for APRO function

COBOL-Name	C/C++-Name	Bedeutung
KCPI	kcpi	APRO: process identification
KCPADM	kcpadm	extention for PADM function
KCACT	kcact	PADM: action
KCADRLT	kcadrlt	PADM: lterm name
KCNORPLY	kcnorply	CTRL: reply message not permitted
KCMCOM	kcmcom	redefinition for MCOM function
KCPOS	kcpos	MCOM: destination in positive case
KCNEG	kcneg	MCOM: destination in negative case
KCCOMID	kccomid	MCOM: complex identification
KCSGCL[1]	kc_sgcl	extensions for SIGN CL
KCLANGID[1]	kclangid	language id of user
KCTERRID[1]	kcterrid	territorial id of user
KCCSNAME[1]	kccsname	character set name of user
KCGTM	kcgtm	creation time of message (generation time)
KCQRC	kcqrc	queue-specific redelivery counter
KCDPID	kcdpid	DPUT ID of message

[1] Nur auf BS2000-Systemen

12.3 ASCII - EBCDIC Code-Konvertierung

Für den Fall, dass die UTM-Anwendung und die Kommunikationspartner mit unterschiedlichen Codes arbeiten (EBCDIC, ASCII-kompatibler Code), bietet UTM die Möglichkeit, per Generierung partnerspezifisch eine automatische Code-Konvertierung zu veranlassen.

Dazu stellt UTM vier Code-Konvertierungen bereit, die die Daten wie folgt konvertieren:

BS2000-, Unix- und Linux-Systeme:

- SYS1/SYS/SYSTEM: ISO8859-i EBCDIC.DF.04.i (EDF04i)
- SYS2: ISO8859-1 EBCDIC.DF.04.DRV (EDF04DRV)
- SYS3: ISO646-IRV EBCDIC.03.DF.03.IRV (EDF03IRV))
- SYS4: ISO646-IRV EBCDIC.03.DF.03.DRV (EDF03DRV).

Windows-Systeme:

- SYS1/SYS/SYSTEM: Windows-1252 EBCDIC.DF.04.F (EDF04F)
- SYS2: Windows-1252 EBCDIC.DF.04.DRV (EDF04DRV)
- SYS3: ISO646-IRV EBCDIC.03.DF.03.IRV (EDF03IRV))
- SYS4: ISO646-IRV EBCDIC.03.DF.03.DRV (EDF03DRV).

SYS1 und SYS2 sind Konvertierungen zwischen zwei 8-Bit-Codes. SYS3 und SYS4 sind Konvertierungen zwischen zwei 7-Bit-Codes.

Die folgenden Abschnitte beschreiben für die einzelnen Plattformen:

- für welche Partner die Konvertierungstabellen verwendet werden dürfen,
- wie Sie sie modifizieren/ersetzen können.

Einen Überblick über die Generierung der Code-Konvertierung entnehmen Sie bitte dem Kapitel "Code-Konvertierung" im openUTM-Handbuch „Anwendungen generieren“.

12.3.1 BS2000-Systeme

Auf BS2000-Systemen kann eine automatische Code-Konvertierung für die Kommunikation der UTM-Anwendung mit einer TS-Anwendung vom Typ SOCKET-USP (Operand MAP= in der PTERM- oder TPOOL- Anweisung) und mit HTTP-Clients (Anweisung CHAR-SET und Operand CONVERT-TEXT der Anweisung HTTP-DESCRIPTOR) generiert werden. Der Grund ist, dass sowohl die Socket-USP-Anwendung als auch HTTP-Clients meist einen ASCII-kompatiblen Code, die UTM-Anwendung hingegen in der Regel einen EBCDIC-Code.

Code-Konvertierungstabellen auf BS2000-Systemen modifizieren

Die Tabellen sind im Assembler Modul KDCEA definiert. Die Source von KDCEA befindet sich in der Bibliothek SYSLIB.UTM.070.EXAMPLE.

Wenn eine Tabelle geändert werden soll, gehen Sie wie folgt vor:

1. Modifizieren Sie die Tabelle im Modul KDCEA.
2. Assemblieren Sie das Modul KDCEA.
3. Binden Sie KDCEA per Anweisung INCLUDE-MODULES zum Anwendungsprogramm dazu, und zwar **bevor** die Anweisung RESOLVE-BY-AUTOLINK auf die Bibliothek SYSLNK.UTM.070 wirksam wird!

12.3.2 Unix-, Linux- und Windows-Systeme

Auf Unix-, Linux- und Windows-Systemen kann eine automatische Code-Konvertierung sowohl für UPIC-Clients und TS-Anwendungen vom Typ APPLI und SOCKET-USP als auch für Server-Server-Kommunikation mit LU6.1- und OSI TP-Partnern generiert werden (Operand MAP= in der PTERM-, TPOOL-, OSI-CON- und SESCHA-Anweisung).

Die Code-Konvertierungstabellen sind in der C-Source `kcsaeea.c` definiert, die sich in folgendem Verzeichnis befindet:

utmpfad`/src/kcsaeea.c` (Unix- und Linux-Systeme)

utmpfad`\src\kcsaeea.c` (Windows-Systeme)

12.3.2.1 Code-Konvertierungstabellen auf Unix- und Linux-Systemen modifizieren

Auf Unix- und Linux-Systemen können Sie diese Tabellen wie folgt modifizieren:

1. Kopieren Sie die Datei `kcsaeea.c` in ein eigenes Dateiverzeichnis.
2. Modifizieren Sie die Konvertierungstabellen nach Ihren Wünschen. Dazu editieren Sie die Datei `kcsaeea.c` mit einem Texteditor. `kcsaeea.c` enthält für jede der vier Code-Konvertierungen jeweils zwei Character Arrays der Länge 256. Dabei dient jeweils ein Array zur ASCII EBCDIC-Konvertierung, das andere zur EBCDIC ASCII-Konvertierung.
3. Übersetzen Sie die modifizierte Source-Datei und erzeugen Sie daraus ein Shared Object.
4. Binden Sie dieses zusätzliche Shared Object zum Anwendungsprogramm.

12.3.2.2 Code-Konvertierungstabellen auf Windows-Systemen modifizieren

Die Code-Konvertierungstabellen sind in der Bibliothek `utmconvt.dll` enthalten. `utmconvt.dll` befindet sind im selben Verzeichnis wie `libwork.dll`.

Sie können diese Konvertierungstabellen Ihren eigenen Bedürfnissen anpassen, indem Sie die ausgelieferte Source-Datei `kcsaeea.c` im Verzeichnis *utmpfad* `\src` ändern und daraus eine modifizierte `utmconvt.dll` erstellen.

> ¡ Zusätzlich enthält *utmpfad* `\src` die Ressourcen-Datei `utmconvt.rc` mit Versions- und Copyright-Informationen. Diese Informationen werden angezeigt, wenn man mit der rechten Maustaste die dll-Datei anklickt und *Eigenschaften* auswählt. Diese Datei muss nicht unbedingt mit eingebunden werden.

Bibliothek utmconvt.dll modifizieren

Zum Modifizieren der Bibliothek `utmconvt.dll` sind folgende Schritte notwendig:

1. Kopieren Sie die Datei `kcsaeea.c` in ein eigenes Dateiverzeichnis.
2. Modifizieren Sie die Konvertierungstabellen nach Ihren Wünschen. Dazu editieren Sie die Datei `kcsaeea.c` mit einem Texteditor. `kcsaeea.c` enthält für jede der vier Code-Konvertierungen jeweils zwei Character Arrays der Länge 256. Dabei dient jeweils ein Array zur ASCII EBCDIC-Konvertierung, das andere zur EBCDIC ASCII-Konvertierung.
3. Rufen Sie das Microsoft Visual Studio auf und gehen Sie wie folgt vor:
   a. Erstellen Sie im Verzeichnis *utmpfad* ein neues Win32- bzw. X64-Projekt mit dem Namen `utmconvt` und dem Anwendungstyp *Dynamic-Link Library*.
   b. Fügen Sie die folgenden Dateien zum Projekt hinzu:
      - die modifizierte Code-Tabellen-Datei `kcsaeea.c`,
      - und gegebenenfalls `utmconvt.rc`.
   c. Weisen Sie die Datei `utmconvt.def` dem Projekt als Moduldefinitionsdatei zu (*Projekteigenschaften Linker Moduldefinitionsdatei*).
   d. Erstellen Sie mit diesem Projekt die Bibliothek `utmconvt.dll`.
   e. Schließen Sie das Microsoft Visual Studio.
4. Ersetzen Sie die alte Bibliothek `utmconvt.dll` durch die neue Bibliothek:
   - Sichern Sie als Erstes die alte Bibliothek unter einem anderen Namen, damit Sie im Fehlerfall wieder darauf zurückgreifen können.
   - Kopieren Sie die neue `utmconvt.dll` in das Verzeichnis, in dem die UTM-Bibliothek `libwork.dll` steht (in der Regel ist dies *utmpfad* `\ex`). Vergewissern Sie sich, dass die ursprüngliche `utmconvt.dll` dabei auch tatsächlich durch die neue `utmconvt.dll` ersetzt wurde.

Die neue Konvertierungsbibliothek ist damit einsatzfähig.

13 Fachwörter

Fachwörter, die an anderer Stelle erklärt werden, sind mit *kursiver* Schrift ausgezeichnet.

Ablaufinvariantes Programm
reentrant program

siehe *reentrant-fähiges Programm*.

Abnormale Beendigung einer UTM-Anwendung
abnormal termination of a UTM application

Beendigung einer *UTM-Anwendung*, bei der die *KDCFILE* nicht mehr aktualisiert wird. Eine abnormale Beendigung wird ausgelöst durch einen schwerwiegenden Fehler, z.B. Rechnerausfall, Fehler in der Systemsoftware. Wird die Anwendung erneut gestartet, führt openUTM einen *Warmstart* durch.

Abstrakte Syntax (OSI)
abstract syntax

Eine abstrakte Syntax ist die Menge der formal beschriebenen Datentypen, die zwischen Anwendungen über *OSI TP* ausgetauscht werden sollen. Eine abstrakte Syntax ist unabhängig von der eingesetzten Hardware und der jeweiligen Programmiersprache.

Access-List
access list

Eine Access-List definiert die Berechtigung für den Zugriff auf einen bestimmten *Service*, auf eine bestimmte *TAC-Queue* oder auf eine bestimmte *USER-Queue*. Eine Access-List ist als *Keyset* definiert und enthält einen oder mehrere *Keycodes*, die jeweils eine Rolle in der Anwendung repräsentieren. Benutzer, LTERMs oder (OSI-)LPAPs dürfen nur dann auf den Service oder die *TAC-Queue*/*USER-Queue* zugreifen, wenn ihnen die entsprechenden Rollen zugeteilt wurden, d.h. wenn ihr *Keyset* und die Access-List mindestens einen gemeinsamen *Keycode* enthalten.

Access Point (OSI)

siehe *Dienstzugriffspunkt.*

ACID-Eigenschaften
ACID properties

Abkürzende Bezeichnung für die grundlegenden Eigenschaften von *Transaktionen*: Atomicity, Consistency, Isolation und Durability.

Administration
administration

Verwaltung und Steuerung einer *UTM-Anwendung* durch einen *Administrator* oder ein *Administrationsprogramm*.

Administrations-Journal
administration journal

siehe *Cluster-Administrations-Journal*.

Administrationskommando
administration command

Kommandos, mit denen der *Administrator* einer *UTM-Anwendung* Administrationsfunktionen für diese Anwendung durchführt. Die Administrationskommandos sind als *Transaktionscodes* realisiert.

Administrationsprogramm
administration program

Teilprogramm, das Aufrufe der *Programmschnittstelle für die Administration* enthält. Dies kann das Standard-Administrationsprogramm *KDCADM* sein, das mit openUTM ausgeliefert wird, oder ein vom Anwender selbst erstelltes Programm.

Administrator
administrator

Benutzer mit Administrationsberechtigung.

AES

AES (Advanced Encryption Standard) ist der aktuelle symmetrische Verschlüsselungsstandard, festgelegt vom NIST (National Institute of Standards and Technology), basierend auf dem an der Universität Leuven (B) entwickelten Rijndael-Algorithmus. Wird das AES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen AES-Schlüssel.

Akzeptor (CPI-C)
acceptor

Die Kommunikationspartner einer *Conversation* werden *Initiator* und Akzeptor genannt. Der Akzeptor nimmt die vom Initiator eingeleitete Conversation mit Accept_Conversation entgegen.

Anmelde-Vorgang (KDCS)
sign-on service

Spezieller *Dialog-Vorgang*, bei dem die Anmeldung eines Benutzers an eine UTM-Anwendung durch *Teilprogramme* gesteuert wird.

Anschlussprogramm
linkage program

siehe *KDCROOT*.

Anwendungsinformation
application information

Sie stellt die Gesamtmenge der von der *UTM-Anwendung* benutzten Daten dar. Dabei handelt es sich um Speicherbereiche und Nachrichten der UTM-Anwendung, einschließlich der aktuell auf dem Bildschirm angezeigten Daten. Arbeitet die UTM-Anwendung koordiniert mit einem Datenbanksystem, so gehören die in der Datenbank gespeicherten Daten ebenfalls zur Anwendungsinformation.

Anwendungs-Kaltstart
application cold start

siehe *Kaltstart*.

Anwendungsprogramm
application program

Ein Anwendungsprogramm bildet den Hauptbestandteil einer *UTM-Anwendung*. Es besteht aus der Main Routine *KDCROOT* und den *Teilprogrammen*. Es bearbeitet alle Aufträge, die an eine *UTM-Anwendung* gerichtet werden.

Anwendungs-Warmstart
application warm start

siehe *Warmstart*.

Apache Axis

Apache Axis (Apache eXtensible Interaction System) ist eine SOAP-Engine zur Konstruktion von darauf basierenden Web Services und Client-Anwendungen. Es existiert eine Implementierung in C++ und Java.

Apache Tomcat

Apache Tomcat stellt eine Umgebung zur Ausführung von Java-Code auf Web-Servern bereit, die im Rahmen des Jakarta-Projekts der Apache Software Foundation entwickelt wird. Es handelt sich um einen in Java geschriebenen Servlet-Container, der mithilfe des JSP-Compilers Jasper auch JavaServer Pages in Servlets übersetzen und ausführen kann. Dazu kommt ein kompletter HTTP-Server.

Application Context (OSI)
application context

Der Application Context ist die Menge der Regeln, die für die Kommunikation zwischen zwei Anwendungen gelten sollen. Dazu gehören z.B. die *abstrakten Syntaxen* und die zugeordneten *Transfer-Syntaxen*.

Application Entity (OSI)
application entity

Eine Application Entity (AE) repräsentiert alle für die Kommunikation relevanten Aspekte einer realen Anwendung. Eine Application Entity wird durch einen global (d.h. weltweit) eindeutigen Namen identifiziert, den *Application Entity Title* (AET). Jede Application Entity repräsentiert genau einen *Application Process*. Ein Application Process kann mehrere Application Entities umfassen.

Application Entity Qualifier (OSI)
application entity qualifier

Bestandteil des *Application Entity Titles*. Der Application Entity Qualifier identifiziert einen *Dienstzugriffspunkt* innerhalb der Anwendung. Ein Application Entity Qualifier kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ "Zahl".

Application Entity Title (OSI)
application entity title

Ein Application Entity Title ist ein global (d.h. weltweit) eindeutiger Name für eine *Application Entity*. Er setzt sich zusammen aus dem *Application Process Title* des jeweiligen *Application Process* und dem *Application Entity Qualifier.*

Application Process (OSI)
application process

Der Application Process repräsentiert im *OSI-Referenzmodell* eine Anwendung. Er wird durch den *Application Process Title* global (d.h. weltweit) eindeutig identifiziert.

Application Process Title (OSI)
application process title

Gemäß der OSI-Norm dient der Application Process Title (APT) zur global (d.h. weltweit) eindeutigen Identifizierung von Anwendungen. Er kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ *Object Identifier*.

Application Service Element (OSI)
application service element

Ein Application Service Element (ASE) repräsentiert eine Funktionsgruppe der Anwendungsschicht (Schicht 7) des *OSI-Referenzmodells*.

Association (OSI)
association

Eine Association ist eine Kommunikationsbeziehung zwischen zwei *Application Entities*. Dem Begriff Association entspricht der *LU6.1*-Begriff *Session*.

Asynchron-Auftrag
queued job

Auftrag, der vom Auftraggeber zeitlich entkoppelt durchgeführt wird. Zur Bearbeitung von Asynchron-Aufträgen sind in openUTM *Message Queuing* Funktionen integriert, vgl. *UTM-gesteuerte Queue* und *Service-gesteuerte Queue*. Ein Asynchron-Auftrag wird durch die *Asynchron-Nachricht*, den Empfänger und ggf. den gewünschten Ausführungszeitpunkt beschrieben.
Ist der Empfänger ein Terminal, ein Drucker oder eine Transportsystem-Anwendung, so ist der Asynchron-Auftrag ein *Ausgabe-Auftrag*; ist der Empfänger ein Asynchron-Vorgang derselben oder einer fernen Anwendung, so handelt es sich um einen *Hintergrund-Auftrag*.
Asynchron-Aufträge können *zeitgesteuerte Aufträge* sein oder auch in einen *Auftrags-Komplex* integriert sein.

Asynchron-Conversation
asynchronous conversation

CPI-C-Conversation, bei der nur der *Initiator* senden darf. Für den *Akzeptor* muss in der *UTM-Anwendung* ein asynchroner Transaktionscode generiert sein.

Asynchron-Nachricht
asynchronous message

Asynchron-Nachrichten sind Nachrichten, die an eine *Message Queue* gerichtet sind. Sie werden von der lokalen *UTM-Anwendung* zunächst zwischengespeichert und dann unabhängig vom Auftraggeber weiter verarbeitet. Je nach Empfänger unterscheidet man folgende Typen von Asynchron-Nachrichten:

- Bei Asynchron-Nachrichten an eine *UTM-gesteuerte Queue* wird die Weiterverarbeitung komplett durch openUTM gesteuert. Zu diesem Typ gehören Nachrichten, die einen lokalen oder fernen *Asynchron-Vorgang* starten (vgl. auch *Hintergrund-Auftrag*) und Nachrichten, die zur Ausgabe an ein Terminal, einen Drucker oder eine Transportsystem-Anwendung geschickt werden (vgl. auch *Ausgabe-Auftrag*).
- Bei Asynchron-Nachrichten an eine *Service-gesteuerte Queue* wird die Weiterverarbeitung durch einen *Service* der Anwendung gesteuert. Zu diesem Typ gehören Nachrichten an eine *TAC-Queue*, Nachrichten an eine *USER-Queue* und Nachrichten an eine *Temporäre Queue*. Die User-Queue und die Temporäre Queue müssen dabei zur lokalen Anwendung gehören, die TAC-Queue kann sowohl in der lokalen als auch in einer fernen Anwendung liegen.

Asynchron-Programm
asynchronous program

Teilprogramm, das von einem *Hintergrund-Auftrag* gestartet wird.

Asynchron-Vorgang (KDCS)
asynchronous service

Vorgang , der einen *Hintergrund-Auftrag* bearbeitet. Die Verarbeitung erfolgt entkoppelt vom Auftraggeber. Ein Asynchron-Vorgang kann aus einem oder mehreren Teilprogrammen /Transaktionen bestehen. Er wird über einen asynchronen *Transaktionscode* gestartet.

Auftrag
job

Anforderung eines *Services* , der von einer *UTM-Anwendung* zur Verfügung gestellt wird, durch Angabe eines *Transaktionscodes* . Siehe auch: *Ausgabe-Auftrag* , *Dialog-Auftrag* , *Hintergrund-Auftrag* , *Auftrags-Komplex*.

Auftraggeber-Vorgang
job-submitting service

Ein Auftraggeber-Vorgang ist ein *Vorgang*, der zur Bearbeitung eines Auftrags einen Service von einer anderen Server-Anwendung (*Auftragnehmer-Vorgang*) anfordert.

Auftragnehmer-Vorgang
job-receiving service

Ein Auftragnehmer-Vorgang ist ein *Vorgang*, der von einem *Auftraggeber-Vorgang* einer anderen Server-Anwendung gestartet wird.

Auftrags-Komplex
job complex

Auftrags-Komplexe dienen dazu, *Asynchron-Aufträgen Quittungsaufträge* zuzuordnen. Ein Asynchron-Auftrag innerhalb eines Auftrags-Komplexes wird *Basis-Auftrag* genannt.

Ausgabe-Auftrag
queued output job

Ausgabeaufträge sind *Asynchron-Aufträge*, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker, ein Terminal oder eine Transportsystem-Anwendung auszugeben. Ausgabeaufträge werden ausschließlich von UTM-Systemfunktionen bearbeitet, d.h. für die Bearbeitung müssen keine Teilprogramme erstellt werden.

Authentisierung
authentication

siehe *Zugangskontrolle*.

Autorisierung
authorization

siehe *Zugriffskontrolle*.

Axis

siehe *Apache Axis*.

Basis-Auftrag
basic job

Asynchron-Auftrag in einem *Auftrags-Komplex*.

Basisformat
basic format

Format, in das der Terminal-Benutzer alle Angaben eintragen kann, die notwendig sind, um einen Vorgang zu starten.

Basisname
filebase

Basisname der UTM-Anwendung.
Auf BS2000-Systemen ist Basisname das Präfix für die *KDCFILE*, die *Benutzerprotokoll-Datei* USLOG und die *System-Protokolldatei* SYSLOG.
Auf Unix-, Linux- und Windows-Systemen ist Basisname der Name des Verzeichnisses, unter dem die KDCFILE, die Benutzerprotokoll-Datei USLOG, die System-Protokolldatei SYSLOG und weitere Dateien der UTM-Anwendung abgelegt sind.

Basisname der Knoten-Anwendung
node filebase

Dateinamens-Präfix bzw. Verzeichnisname für die *KDCFILE*, *Benutzerprotokoll-Datei* und *Systemprotokoll-Datei* der *Knoten-Anwendung*.

Basisname der UTM-Cluster-Anwendung
cluster filebase

Dateinamens-Präfix bzw. Verzeichnisname für die *UTM-Cluster-Dateien*.

Benutzerausgang
user exit

Begriff ersetzt durch *Event-Exit.*

Benutzerkennung
user ID

Bezeichner für einen Benutzer, der in der *Konfiguration* der *UTM-Anwendung* festgelegt ist (optional mit Passwort zur *Zugangskontrolle*) und dem spezielle Zugriffsrechte (*Zugriffskontrolle*) zugeordnet sind. Ein Terminal-Benutzer muss bei der Anmeldung an die UTM-Anwendung diesen Bezeichner (und ggf. das zugeordnete Passwort) angeben. Auf BS2000-Systemen ist außerdem eine Zugangskontrolle über *Kerberos* möglich.
Für andere Clients ist die Angabe der Benutzerkennung optional, siehe auch *Verbindungs-Benutzerkennung*.
UTM-Anwendungen können auch ohne Benutzerkennungen generiert werden.

Benutzer-Protokolldatei
user log file

Datei oder Dateigeneration, in die der Benutzer mit dem KDCS-Aufruf LPUT Sätze variabler Länge schreibt. Jedem Satz werden die Daten aus dem KB-Kopf des *KDCS-Kommunikationsbereichs* vorangestellt. Die Benutzerprotokolldatei unterliegt der Transaktionssicherung von openUTM.

Berechtigungsprüfung
sign-on check

siehe *Zugangskontrolle*.

Beweissicherung (BS2000-Systeme)
audit

Im Betrieb einer *UTM-Anwendung* können zur Beweissicherung sicherheitsrelevante UTM-Ereignisse von *SAT* protokolliert werden.

Bildschirm-Wiederanlauf
screen restart

Wird ein *Dialog-Vorgang* unterbrochen, gibt openUTM beim *Vorgangswiederanlauf* die *Dialog-Nachricht* der letzten abgeschlossenen *Transaktion* erneut auf dem Bildschirm aus, sofern die letzte Transaktion eine Nachricht auf den Bildschirm ausgegeben hat.

Browsen von Asynchron-Nachrichten
browsing asynchronous messages

Ein *Vorgang* liest nacheinander die *Asynchron-Nachrichten*, die sich in einer *Service-gesteuerten Queue* befinden. Die Nachrichten werden während des Lesens nicht gesperrt und verbleiben nach dem Lesen in der Queue. Dadurch ist gleichzeitiges Lesen durch unterschiedliche Vorgänge möglich.

Bypass-Betrieb (BS2000-Systeme)
bypass mode

Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Im Bypass-Betrieb wird eine an den Drucker gerichtete *Asynchron-Nachricht* an das Terminal gesendet und von diesem auf den Drucker umgeleitet, ohne auf dem Bildschirm angezeigt zu werden.

Cache-Speicher
cache

Pufferbereich zur Zwischenspeicherung von Anwenderdaten für alle Prozesse einer *UTM-Anwendung*. Der Cache-Speicher dient zur Optimierung der Zugriffe auf den *Pagepool* und für UTM-Cluster-Anwendungen zusätzlich auf den *Cluster-Pagepool*.

CCR (Commitment, Concurrency and Recovery)

CCR ist ein von OSI definiertes Application Service Element (ASE) für die OSI-TP-Kommunkation, welches die Protokollelemente (Services) zum Beginn und Abschluss (Commit oder Rollback) einer *Transaktion* enthält. CCR unterstützt das Zwei-Phasen-Commitment.

CCS-Name (BS2000-Systeme)
CCS name

siehe *Coded-Character-Set-Name* .

Client
client

Clients einer *UTM-Anwendung* können sein:

- Terminals
- UPIC-Client-Programme
- Transportsystem-Anwendungen (z.B. DCAM-, PDN-, CMX-, Socket-Anwendungen oder UTM-Anwendungen, die als *Transportsystem-Anwendung* generiert sind)

Clients werden über LTERM-Partner an die UTM-Anwendung angeschlossen.
Hinweis: UTM-Clients mit Trägersystem OpenCPIC werden wie *OSI TP-Partner* behandelt.

Client-Seite einer Conversation
client side of a conversation

Begriff ersetzt durch *Initiator*.

Cluster

Eine Anzahl von Rechnern, die über ein schnelles Netzwerk verbunden sind und die von außen in vielen Fällen als ein Rechner gesehen werden können. Das Ziel des "Clustering" ist meist die Erhöhung der Rechenkapazität oder der Verfügbarkeit gegenüber einem einzelnen Rechner.

Cluster-Administrations-Journal
cluster administration journal

Das Cluster-Administrations-Journal besteht aus:

- zwei Protokolldateien mit Endungen JRN1 und JRN2 für globale Administrationsaktionen,
- der JKAA-Datei, die eine Kopie der KDCS Application Area (KAA) enthält. Aus dieser Kopie werden administrative Änderungen übernommen, die nicht mehr in den beiden Protokolldateien enthalten sind.

Die Administrations-Journal-Dateien dienen dazu, administrative Aktionen, die in einer UTM-Cluster-Anwendung Cluster-weit auf alle Knoten-Anwendungen wirken sollen, an die anderen Knoten-Anwendungen weiterzugeben.

Cluster-GSSB-Datei
cluster GSSB file

Datei zur Verwaltung von GSSBs in einer *UTM-Cluster-Anwendung*. Die Cluster-GSSB-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Konfigurationsdatei
cluster configuration file

Datei, die die zentralen Konfigurationsdaten einer *UTM-Cluster-Anwendung* enthält. Die Cluster-Konfigurationsdatei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Lock-Datei
cluster lock file

Datei einer *UTM-Cluster-Anwendung*, die dazu dient, Knoten-übergreifende Sperren auf Anwenderdatenbereiche zu verwalten.

Cluster-Pagepool
cluster pagepool

Der Cluster-Pagepool besteht aus einer Verwaltungsdatei und bis zu 10 Dateien, in denen die Cluster-weit verfügbaren Anwenderdaten (Vorgangsdaten inklusive LSSB, GSSB und ULS) einer *UTM-Cluster-Anwendung* gespeichert werden. Der Cluster-Pagepool wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Startserialisierungs-Datei
cluster start serialization file

Lock-Datei, mit der die Starts einzelner Knoten-Anwendungen serialisiert werden (nur auf Unix-, Linux- und Windows-Systemen).

Cluster-ULS-Datei
cluster ULS file

Datei zur Verwaltung von ULS-Bereichen einer *UTM-Cluster-Anwendung*. Die Cluster-ULS-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-User-Datei
cluster user file

Datei, die die Verwaltungsdaten der Benutzer einer *UTM-Cluster-Anwendung* enthält. Die Cluster-User-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Coded-Character-Set-Name (BS2000-Systeme)
coded character set name

Bei Verwendung des Produkts *XHCS* (e**X**tended **H**ost **C**ode **S**upport) wird jeder verwendete Zeichensatz durch einen Coded-Character-Set-Namen (abgekürzt: "CCS-Name" oder "CCSN") eindeutig identifiziert.

Communication Resource Manager
communication resource manager

Communication Resource Manager (CRMs) kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen. openUTM stellt CRMs für den internationalen Standard OSI TP, für den Industrie-Standard *LU6.1* und für das openUTM-eigene Protokoll UPIC zur Verfügung.

Contention Loser
contention loser

Jede Verbindung zwischen zwei Partnern wird von einem der Partner verwaltet. Der Partner, der die Verbindung verwaltet, heißt *Contention Winner*. Der andere Partner ist der Contention Loser.

Contention Winner
contention winner

Der Contention Winner einer Verbindung übernimmt die Verwaltung der Verbindung. Aufträge können sowohl vom Contention Winner als auch vom *Contention Loser* gestartet werden. Im Konfliktfall, wenn beide Kommunikationspartner gleichzeitig einen Auftrag starten wollen, wird die Verbindung vom Auftrag des Contention Winner belegt.

Conversation
conversation

Bei CPI-C nennt man die Kommunikation zwischen zwei CPI-C-Anwendungsprogrammen Conversation. Die Kommunikationspartner einer Conversation werden *Initiator* und *Akzeptor* genannt.

Conversation-ID
conversation ID

Jeder *Conversation* wird von CPI-C lokal eine Conversation-ID zugeordnet, d.h. *Initiator* und *Akzeptor* haben jeweils eine eigene Conversation-ID. Mit der Conversation-ID wird jeder CPI-C-Aufruf innerhalb eines Programms eindeutig einer Conversation zugeordnet.

CPI-C

CPI-C (**C**ommon **P**rogramming **I**nterface for **C**ommunication) ist eine von X/Open und dem CIW (**C**PI-C **I**mplementor's **W**orkshop) normierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen. Das in openUTM implementierte CPI-C genügt der CPI-C V2.0 CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung.
CPI-C in openUTM kann über die Protokolle OSI TP, LU6.1, UPIC und mit openUTM-LU6.2 kommunizieren.

Cross Coupled System / XCS

Verbund von BS2000-Rechnern mit *Highly Integrated System Complex* Multiple System Control Facility (HIPLEX® MSCF).

Datenraum (BS2000-Systeme)
data space

Virtueller Adressraum des BS2000, der in seiner gesamten Größe vom Anwender genutzt werden kann.
In einem Datenraum können nur Daten und als Daten abgelegte Programme adressiert werden, es kann kein Programmcode zum Ablauf gebracht werden.

Dead Letter Queue
dead letter queue

Die Dead Letter Queue ist eine *TAC-Queue* mit dem festen Namen KDCDLETQ. Sie steht immer zur Verfügung, um Asynchron-Nachrichten an *Transaktionscodes*, TAC-Queues, LPAP- oder OSI-LPAP-Partner zu sichern, die nicht verarbeitet werden konnten.
Die Sicherung von Asynchron-Nachrichten in der Dead Letter Queue kann durch den Parameter DEAD-LETTER-Q der TAC-, LPAP- oder OSI-LPAP-Anweisung für jedes Nachrichtenziel einzeln ein- und ausgeschaltet werden.

DES

DES (Data Encryption Standard) ist eine internationale Norm zur Verschlüsselung von Daten. Bei diesem Verfahren wird ein Schlüssel zum Ver- und Entschlüsseln verwendet. Wird das DES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen DES-Schlüssel.

Dialog-Auftrag
dialog job, interactive job

Auftrag, der einen *Dialog-Vorgang* startet. Der Auftrag kann von einem *Client* oder - bei *Server-Server-Kommunikation* - von einer anderen Anwendung erteilt werden.

Dialog-Conversation
dialog conversation

CPI-C-Conversation, bei der sowohl der *Initiator* als auch der *Akzeptor* senden darf. Für den *Akzeptor* muss in der *UTM-Anwendung* ein Dialog-Transaktionscode generiert sein.

Dialog-Nachricht
dialog message

Nachricht, die eine Antwort erfordert oder selbst eine Antwort auf eine Anfrage ist. Dabei bilden Anfrage und Antwort einen *Dialog-Schritt*.

Dialog-Programm
dialog program

Teilprogramm, das einen *Dialog-Schritt* teilweise oder vollständig bearbeitet.

Dialog-Schritt
dialog step

Ein Dialog-Schritt beginnt mit dem Empfang einer *Dialog-Nachricht* durch die *UTM-Anwendung*. Er endet mit der Antwort der UTM-Anwendung.

Dialog-Terminalprozess (Unix-, Linux- und Windows-Systeme)
dialog terminal process

Ein Dialog-Terminalprozess verbindet ein Unix-, Linux- oder Windows-Terminal mit den *Workprozessen* der *UTM-Anwendung*. Dialog-Terminalprozesse werden entweder vom Benutzer durch Eingabe von utmdtp oder über die LOGIN-Shell gestartet. Für jedes Terminal, das an eine UTM-Anwendung angeschlossen werden soll, ist ein eigener Dialog-Terminalprozess erforderlich.

Dialog-Vorgang
dialog service

Vorgang , der einen *Auftrag* im Dialog (zeitlich gekoppelt) mit dem Auftraggeber (*Client* oder eine andere Server-Anwendung) bearbeitet. Ein Dialog-Vorgang verarbeitet *Dialog-Nachrichten* vom Auftraggeber und erzeugt Dialog-Nachrichten für diesen. Ein Dialog-Vorgang besteht aus mindestens einer *Transaktion.* Ein Dialog-Vorgang umfasst in der Regel mindestens einen *Dialog-Schritt*. Ausnahme: Bei *Vorgangskettung* können auch mehrere Vorgänge einen Dialog-Schritt bilden.

Dienst
service

Programm auf Windows-Systemen, das im Hintergrund unabhängig von angemeldeten Benutzern oder Fenstern abläuft.

Dienstzugriffspunkt
service access point

Im *OSI-Referenzmodell* stehen einer Schicht am Dienstzugriffspunkt die Leistungen der darunterliegenden Schicht zur Verfügung. Der Dienstzugriffspunkt wird im lokalen System durch einen *Selektor* identifiziert. Bei der Kommunikation bindet sich die *UTM-Anwendung* an einen Dienstzugriffspunkt. Eine Verbindung wird zwischen zwei Dienstzugriffspunkten aufgebaut.

Distributed Transaction Processing

X/Open-Architekturmodell für die transaktionsorientierte *verteilte Verarbeitung*.

Druckadministration
print administration

Funktionen zur *Drucksteuerung* und Administration von *Ausgabeaufträgen*, die an einen Drucker gerichtet sind.

Druckerbündel
printer pool

Mehrere Drucker, die demselben *LTERM-Partner* zugeordnet sind.

Druckergruppe (Unix- und Linux-Systeme)
printer group

Die Unix- oder Linux-Plattform richtet für jeden Drucker standardmäßig eine Druckergruppe ein, die genau diesen Drucker enthält. Darüber hinaus lassen sich mehrere Drucker einer Druckergruppe, aber auch ein Drucker mehreren Druckergruppen zuordnen.

Druckerprozess (Unix- und Linux-Systeme)
printer process

Prozess, der vom *Mainprozess* zur Ausgabe von *Asynchron-Nachrichten* an eine *Druckergruppe* eingerichtet wird. Er existiert, solange die Druckergruppe an die *UTM-Anwendung* angeschlossen ist. Pro angeschlossener Druckergruppe gibt es einen Druckerprozess.

Druckersteuerstation
printer control terminal

Begriff wurde ersetzt durch *Druckersteuer-LTERM*.

Druckersteuer-LTERM
printer control LTERM

Über ein Druckersteuer-LTERM kann sich ein *Client* oder ein Terminal-Benutzer an eine *UTM-Anwendung* anschließen. Von dem Client-Programm oder Terminal aus kann dann die *Administration* der Drucker erfolgen, die dem Druckersteuer-LTERM zugeordnet sind. Hierfür ist keine Administrationsberechtigung notwendig.

Drucksteuerung
print control

openUTM-Funktionen zur Steuerung von Druckausgaben.

Dynamische Konfiguration
dynamic configuration

Änderung der *Konfiguration* durch die Administration. Im laufenden Betrieb der Anwendung können UTM-Objekte wie z.B. *Teilprogramme*, *Transaktionscodes*, *Clients*, *LU6.1-Verbindungen*, Drucker oder *Benutzerkennungen* in die Konfiguration aufgenommen, modifiziert oder teilweise auch gelöscht werden. Hierzu können die Administrationsprogramme WinAdmin oder WebAdmin verwendet werden, oder es müssen eigene *Administrationsprogramme* erstellt werden, die die Funktionen der *Programmschnittstelle der Administration* nutzen.

Einschritt-Transaktion
single-step transaction

Transaktion, die genau einen *Dialog-Schritt* umfasst.

Einschritt-Vorgang
single-step service

Dialog-Vorgang, der genau einen *Dialog-Schritt* umfasst.

Ereignisgesteuerter Vorgang
event-driven service

Begriff ersetzt durch *Event-Service*.

Event-Exit
event exit

Routine des *Anwendungsprogramms*, das bei bestimmten Ereignissen (z.B. Start eines Prozesses, Ende eines Vorgangs) automatisch gestartet wird. Diese darf - im Gegensatz zu den *Event-Services* - keine KDCS-, CPI-C- und XATMI-Aufrufe enthalten.

Event-Funktion
event function

Oberbegriff für *Event-Exits* und *Event-Services*.

Event-Service
event service

Vorgang, der beim Auftreten bestimmter Ereignisse gestartet wird, z.B. bei bestimmten UTM-Meldungen. Die *Teilprogramme* ereignisgesteuerter Vorgänge müssen KDCS-Aufrufe enthalten.

Funktionseinheit, Functional Unit (FU)
functional unit

Teilmenge des *OSI-TP*-Protokolls, die eine bestimmte Funktionalität beinhaltet. Das OSI-TP-Protokoll ist in folgende Funktionseinheiten aufgeteilt:

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

Ein Hersteller, der OSI-TP implementiert, muss nicht alle Funktionseinheiten realisieren, sondern kann sich auf eine Teilmenge beschränken. Eine Kommunikation zwischen Anwendungen zweier unterschiedlicher OSI-TP-Implementierungen ist nur dann möglich, wenn die realisierten Funktionseinheiten zueinander passen.

Generierung
generation

siehe *UTM-Generierung*.

Globaler Sekundärer Speicherbereich/GSSB
global secondary storage area

siehe *Sekundärspeicherbereich.*

Hardcopy-Betrieb
hardcopy mode

Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Dabei wird eine Nachricht, die auf dem Bildschirm angezeigt wird, zusätzlich auf dem Drucker abgedruckt.

Heterogene Kopplung
heterogeneous link

Bei *Server-Server-Kommunikation:* Kopplung einer *UTM-Anwendung* mit einer Nicht-UTM-Anwendung, z.B. einer CICS- oder TUXEDO-Anwendung.

Highly Integrated System Complex / HIPLEX ®

Produktfamilie zur Realisierung eines Bedien-, Last- und Verfügbarkeitsverbunds mit mehreren BS2000-Servern.

Hintergrund-Auftrag
background job

Hintergrund-Aufträge sind *Asynchron-Aufträge*, die an einen *Asynchron-Vorgang* der eigenen oder einer fernen Anwendung gerichtet sind. Hintergrund-Aufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluss auf den aktuellen Dialog hat.

HIPLEX ® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)
stellt bei HIPLEX® die Infrastruktur sowie Basisfunktionen für verteilte Anwendungen bereit.

Homogene Kopplung
homogeneous link

Bei *Server-Server-Kommunikation*: Kopplung von *UTM-Anwendungen.* Dabei spielt es keine Rolle, ob die Anwendungen auf der gleichen oder auf unterschiedlichen Betriebssystem-Plattformen ablaufen.

Inbound-Conversation (CPI-C)
inbound conversation

siehe *Incoming-Conversation.*

Incoming-Conversation (CPI-C)
incoming conversation

Eine *Conversation*, bei der das lokale CPI-C-Programm *Akzeptor* ist, heißt Incoming-Conversation. In der X/Open-Specification wird für Incoming-Conversation auch das Synonym Inbound-Conversation verwendet.

Initiale KDCFILE
initial KDCFILE

In einer *UTM-Cluster-Anwendung* die *KDCFILE*, die von *KDCDEF* erzeugt wurde und vor dem Start der Knoten-Anwendungen für jeden Knoten kopiert werden muss.

Initiator (CPI-C)
initiator

Die Kommunikationspartner einer *Conversation* werden Initiator und *Akzeptor* genannt. Der Initiator baut die Conversation mit den CPI-C-Aufrufen Initialize_Conversation und Allocate auf.

Insert
insert

Feld in einem Meldungstext, in das openUTM aktuelle Werte einträgt.

Inverser KDCDEF
inverse KDCDEF

Funktion, die aus den Konfigurationsdaten der *KDCFILE*, die im laufenden Betrieb dynamisch angepasst wurde, Steueranweisungen für einen *KDCDEF*-Lauf erzeugt. Der inverse KDCDEF kann "offline" unter KDCDEF oder "online" über die *Programmschnittstelle zur Administration* gestartet werden.

IUTMDB
IUTMDB

Schnittstelle für die koordinierte Zusammenarbeit mit externen Resource Managern auf BS2000-Systemen. Dazu gehören Datenhaltungssysteme (LEASY) und Datenbanksysteme (SESAM/SQL, UDS/SQL).

JConnect-Client
JConnect client

Bezeichung für Clients auf Basis des Produkts openUTM-JConnect. Die Kommunikation mit der UTM-Anwendung erfolgt über das *UPIC-Protokoll*.

JDK

Java Development Kit
Standard-Entwicklungsumgebung von Oracle Corporation für die Entwicklung von Java-Anwendungen.

Kaltstart
cold start

Starten einer *UTM-Anwendung* nach einer *normalen Beendigung* der Anwendung oder nach einer Neugenerierung (vgl. auch *Warmstart*).

KDCADM

Standard-Administrationsprogramm, das zusammen mit openUTM ausgeliefert wird. KDCADM stellt Administrationsfunktionen zur Verfügung, die über Transaktionscodes (*Administrationskommandos*) aufgerufen werden.

KDCDEF

UTM-Tool für die *Generierung* von *UTM-Anwendungen*. KDCDEF erstellt anhand der Konfigurationsinformationen in den KDCDEF-Steueranweisungen die UTM-Objekte *KDCFILE* und die ROOT-Tabellen-Source für die Main Routine *KDCROOT*.
In UTM-Cluster-Anwendungen erstellt KDCDEF zusätzlich die *Cluster-Konfigurationsdatei,* die *Cluster-User-Datei*, den *Cluster-Pagepool*, die *Cluster-GSSB-Datei* und die *Cluster-ULS-Datei*.

KDCFILE

Eine oder mehrere Dateien, die für den Ablauf einer *UTM-Anwendung* notwendige Daten enthalten. Die KDCFILE wird mit dem UTM-Generierungstool *KDCDEF* erstellt. Die KDCFILE enthält unter anderem die *Konfiguration* der Anwendung.

KDCROOT

Main Routine eines *Anwendungsprogramms*, die das Bindeglied zwischen *Teilprogrammen* und UTM-Systemcode bildet. KDCROOT wird zusammen mit den *Teilprogrammen* zum *Anwendungsprogramm* gebunden.

KDCS-Parameterbereich
KDCS parameter area

siehe *Parameterbereich.*

KDCS-Programmschnittstelle
KDCS program interface

Universelle UTM-Programmschnittstelle, die den nationalen Standard DIN 66 265 erfüllt und Erweiterungen enthält. Mit KDCS (Kompatible Datenkommunikationsschnittstelle) lassen sich z.B. Dialog-Services erstellen und *Message Queuing* Funktionen nutzen. Außerdem stellt KDCS Aufrufe zur *verteilten Verarbeitung* zur Verfügung.

Kerberos

Kerberos ist ein standardisiertes Netzwerk-Authentisierungsprotokoll (RFC1510), das auf kryptographischen Verschlüsselungsverfahren basiert, wobei keine Passwörter im Klartext über das Netzwerk gesendet werden.

Kerberos-Principal
Kerberos principal

Eigentümer eines Schlüssels.
Kerberos arbeitet mit symmetrischer Verschlüsselung, d.h. alle Schlüssel liegen an zwei Stellen vor, beim Eigentümer eines Schlüssels (Principal) und beim KDC (Key Distribution Center).

Keycode
key code

Code, der in einer Anwendung eine bestimmte Zugriffsberechtigung oder eine bestimmte Rolle repräsentiert. Mehrere Keycodes werden zu einem *Keyset* zusammengefasst.

Keyset
key set

Zusammenfassung von einem oder mehrerer *Keycodes* unter einem bestimmten Namen. Ein Keyset definiert Berechtigungen im Rahmen des verwendeten Berechtigungskonzepts (Lock-/Keycode-Konzept oder *Access-List* -Konzept).
Ein Keyset kann einer *Benutzerkennung* , einem *LTERM-Partner* , einem *(OSI-) LPAP-Partner* , einem *Service* oder einer *TAC-Queue* zugeordnet werden.

Knoten
node

Einzelner Rechner eines *Clusters*.

Knoten-Anwendung
node application

UTM-Anwendung, die als Teil einer *UTM-Cluster-Anwendung* auf einem einzelnen *Knoten* zum Ablauf kommt.

Knoten-Recovery
node recovery

Wenn für eine abnormal beendete Knoten-Anwendung zeitnah kein Warmstart auf ihrem eigenen *Knoten-Rechner* möglich ist, kann man für diesen Knoten auf einem anderen Knoten des UTM-Clusters eine Knoten-Recovery (Wiederherstellung) durchführen. Dadurch können Sperren, die von der ausgefallenen Knoten-Anwendung gehalten werden, freigegeben werden, um die laufende *UTM-Cluster-Anwendung* nicht unnötig zu beeinträchtigen.

Knotengebundener Vorgang
node bound service

Ein knotengebundener Vorgang eines Benutzers kann nur an der Knoten-Anwendung fortgesetzt werden, an der der Benutzer zuletzt angemeldet war. Folgende Vorgänge sind immer knotengebunden:

- Vorgänge, die eine Kommunikation mit einem Auftragnehmer über LU6.1 oder OSI TP begonnen haben und bei denen der Auftragnehmervorgang noch nicht beendet wurde
- eingeschobene Vorgänge einer Vorgangskellerung
- Vorgänge, die eine SESAM-Transaktion abgeschlossen haben

Außerdem ist der Vorgang eines Benutzers knotengebunden, solange der Benutzer an einer Knoten-Anwendung angemeldet ist.

Kommunikationsbereich/KB (KDCS)
communication area

Transaktionsgesicherter KDCS-*Primärspeicherbereich*, der Vorgangs-spezifische Daten enthält. Der Kommunikationsbereich besteht aus 3 Teilen:

- dem KB-Kopf mit allgemeinen Vorgangsdaten
- dem KB-Rückgabebereich für Rückgaben nach KDCS-Aufrufen
- dem KB-Programmbereich zur Datenübergabe zwischen UTM-Teilprogrammen innerhalb eines *Vorgangs*.

Kommunikationsendpunkt
communication end point

siehe *Transportsystem-Endpunkt*

Konfiguration
configuration

Summe aller Eigenschaften einer *UTM-Anwendung*. Die Konfiguration beschreibt:

- Anwendungs- und Betriebsparameter
- die Objekte der Anwendung und die Eigenschaften dieser Objekte. Objekte sind z.B. *Teilprogramme* und *Transaktionscodes*, Kommunikationspartner, Drucker, *Benutzerkennungen*
- definierte Zugriffsschutz- und Zugangsschutzmaßnahmen

Die Konfiguration einer UTM-Anwendung wird bei der UTM-Generierung festgelegt (*statische Konfiguration*) und kann per *Administration* dynamisch (während des Anwendungslaufs) geändert werden (*dynamische Konfiguration*). Die Konfiguration ist in der *KDCFILE* abgelegt.

Logging-Prozess
logging process

Prozess auf Unix-, Linux- und Windows-Systemen, der die Protokollierung von Abrechnungssätzen oder Messdaten steuert.

Logische Verbindung
virtual connection

Zuordnung zweier Kommunikationspartner.

Log4j

Log4j ist ein Teil des Apache Jakarta Projekts. Log4j bietet Schnittstellen zum Protokollieren von Informationen (Ablauf-Informationen, Trace-Records,...) und zum Konfigurieren der Protokoll-Ausgabe. *WS4UTM* verwendet das Softwareprodukt Log4j für die Trace- und Logging-Funktionalität.

Lockcode

Code, um einen LTERM-Partner oder einen Transaktionscode vor unberechtigtem Zugriff zu schützen. Damit ist ein Zugriff nur möglich, wenn das *Keyset* des Zugreifenden den passenden *Keycode* enthält (Lock-/Keycode-Konzept).

Lokaler Sekundärer Speicherbereich/LSSB
local secondary storage area

siehe *Sekundärspeicherbereich.*

LPAP-Bündel
LPAP bundle

LPAP-Bündel ermöglichen die Verteilung von Nachrichten an LPAP-Partner auf mehrere Partner-Anwendungen. Soll eine UTM-Anwendung sehr viele Nachrichten mit einer Partner-Anwendung austauschen, kann es für die Lastverteilung sinnvoll sein, mehrere Instanzen der Partner-Anwendung zu starten und die Nachrichten auf die einzelnen Instanzen zu verteilen. In einem LPAP-Bündel übernimmt openUTM die Verteilung der Nachrichten an die Instanzen der Partner-Anwendung. Ein LPAP-Bündel besteht aus einem Master-LPAP und mehreren Slave-LPAPs. Die Slave-LPAPs werden dem Master-LPAP bei der UTM-Generierung zugeordnet. LPAP-Bündel gibt es sowohl für das OSI TP-Protokoll als auch für das LU6.1-Protokoll.

LPAP-Partner
LPAP partner

Für die *verteilte Verarbeitung* über das *LU6.1*-Protokoll muss in der lokalen Anwendung für jede Partner-Anwendung ein LPAP-Partner konfiguriert werden. Der LPAP-Partner spiegelt in der lokalen Anwendung die Partner-Anwendung wider. Bei der Kommunikation wird die Partner-Anwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten LPAP-Partners angesprochen.

LTERM-Bündel
LTERM bundle

Ein LTERM-Bündel (Verbindungsbündel) besteht aus einem Master-LTERM und mehreren Slave-LTERMs. Mit einem LTERM-Bündel (Verbindungsbündel) verteilen Sie asynchrone Nachrichten an eine logische Partner-Anwendung gleichmäßig auf mehrere parallele Verbindungen.

LTERM-Gruppe
LTERM group

Eine LTERM-Gruppe besteht aus einem oder mehreren Alias-LTERMs, den Gruppen-LTERMs, und einem Primary-LTERM. In einer LTERM-Gruppe ordnen Sie mehrere LTERMs einer Verbindung zu.

LTERM-Partner
LTERM partner

Um *Clients* oder Drucker an eine *UTM-Anwendung* anschließen zu können, müssen in der Anwendung LTERM-Partner konfiguriert werden. Ein Client oder Drucker kann nur angeschlossen werden, wenn ihm ein LTERM-Partner mit entsprechenden Eigenschaften zugeordnet ist. Diese Zuordnung wird i.A. in der *Konfiguration* festgelegt, sie kann aber auch dynamisch über Terminal-Pools erfolgen.

LTERM-Pool
LTERM pool

Statt für jeden *Client* eine LTERM- und eine PTERM-Anweisung anzugeben, kann mit der Anweisung TPOOL ein Pool von LTERM-Partnern definiert werden. Schließt sich ein Client über einen LTERM-Pool an, wird ihm dynamisch ein LTERM-Partner aus dem Pool zugeordnet.

LU6.1

Geräteunabhängiges Datenaustauschprotokoll (Industrie-Standard) für die transaktionsgesicherte *Server-Server-Kommunikation*.

LU6.1-LPAP-Bündel
LU6.1-LPAP bundle

LPAP-Bündel für *LU6.1*-Partner-Anwendungen.

LU6.1-Partner
LU6.1 partner

Partner der *UTM-Anwendung*, der mit der UTM-Anwendung über das Protokoll *LU6.1* kommuniziert. Beispiele für solche Partner sind:

- eine UTM-Anwendung, die über LU6.1 kommuniziert
- eine Anwendung im IBM-Umfeld (z.B. CICS, IMS oder TXSeries), die über LU6.1 kommuniziert

Mainprozess (Unix-, Linux- und Windows-Systeme)
main process

Prozess, der die *UTM-Anwendung* startet. Er startet die *Workprozesse*, die *UTM-System-Prozesse*, *Druckerprozesse, Netzprozesse, Logging-Prozess* und den *Timerprozess* und überwacht die *UTM-Anwendung*.

Main Routine KDCROOT
main routine KDCROOT

siehe *KDCROOT*.

Management Unit
management unit

Komponente des *SE Servers*; ermöglicht mit Hilfe des *SE Managers* ein zentrales, web-basiertes Management aller Units eines SE Servers.

Meldung / UTM-Meldung
UTM message

Meldungen werden vom Transaktionsmonitor openUTM oder von UTM-Tools (wie z.B. *KDCDEF*) an *Meldungsziele* ausgegeben. Eine Meldung besteht aus einer Meldungsnummer und dem Meldungstext, der ggf. *Inserts* mit aktuellen Werten enthält. Je nach Meldungsziel werden entweder die gesamte Meldung oder nur Teile der Meldung (z.B. nur die Inserts) ausgegeben.

Meldungsdefinitionsdatei
message definition file

Die Meldungsdefinitionsdatei wird mit openUTM ausgeliefert und enthält standardmäßig die UTM-Meldungstexte in deutscher und englischer Sprache und die Definitionen der Meldungseigenschaften. Aufbauend auf diese Datei kann der Anwender auch eigene, individuelle Meldungsmodule erzeugen.

Meldungsziel
message destination

Ausgabemedium für eine *Meldung*. Mögliche Meldungsziele von Meldungen des Transaktionsmonitors openUTM sind z.B. Terminals, *TS-Anwendungen*, der *Event-Service* MSGTAC, die *System-Protokolldatei* SYSLOG oder *TAC-Queues, Asynchron-TACs, USER-Queues*, SYSOUT/SYSLST bzw. stderr/stdout. Meldungsziele von Meldungen der UTM-Tools sind SYSOUT /SYSLST bzw. stderr/stdout.

Mehrschritt-Transaktion
multi-step transaction

Transaktion, die aus mehr als einem *Verarbeitungsschritt* besteht.

Mehrschritt-Vorgang (KDCS)
multi-step service

Vorgang, der in mehreren *Dialog-Schritten* ausgeführt wird.

Message Queuing
message queuing

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete *Message Queues* ausgetauscht werden. Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen. Die Übermittlung der Nachricht hängt nicht davon ab, ob gerade eine Netzverbindung besteht oder nicht. Bei openUTM gibt es *UTM-gesteuerte Queues* und *Service-gesteuerte Queues*.

Message Queue
message queue

Warteschlange, in der bestimmte Nachrichten transaktionsgesichert bis zur Weiterverarbeitung eingereiht werden. Je nachdem, wer die Weiterverarbeitung kontrolliert, unterscheidet man *Service-gesteuerte Queues* und *UTM-gesteuerte Queues*.

MSGTAC
MSGTAC

Spezieller Event-Service, der Meldungen mit dem Meldungsziel MSGTAC per Programm verarbeitet. MSGTAC ist ein Asynchron-Vorgang und wird vom Betreiber der Anwendung erstellt.

Multiplex-Verbindung (BS2000-Systeme)
multiplex connection

Spezielle Möglichkeit, die *OMNIS* bietet, um Terminals an eine *UTM-Anwendung* anzuschließen. Eine Multiplex-Verbindung ermöglicht es, dass sich mehrere Terminals eine *Transportverbindung* teilen.

Nachrichten-Bereich/NB (KDCS)
KDCS message area

Bei KDCS-Aufrufen: Puffer-Bereich, in dem Nachrichten oder Daten für openUTM oder für das *Teilprogramm* bereitgestellt werden.

Network File System/Service / NFS

Ermöglicht den Zugriff von Unix- und Linux-Rechnern auf Dateisysteme über das Netzwerk.

Netzprozess (Unix-, Linux- und Windows-Systeme)
net process

Prozess einer *UTM-Anwendung* zur Netzanbindung.

Netzwerk-Selektor
network selector

Der Netzwerk-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Vermittlungsschicht des *OSI-Referenzmodells*.

Normale Beendigung einer UTM-Anwendung
normal termination of a UTM application

Kontrollierte Beendigung einer *UTM-Anwendung*; das bedeutet u.a., dass die Verwaltungsdaten auf der *KDCFILE* aktualisiert werden. Eine normale Beendigung veranlasst der *Administrator* (z.B. mit KDCSHUT N). Den Start nach einer normalen Beendigung führt openUTM als *Kaltstart* durch.

Object Identifier
object identifier

Ein Object Identifier ist ein weltweit eindeutiger Bezeichner für Objekte im OSI-Umfeld. Ein Object Identifier besteht aus einer Folge von ganzen Zahlen, die einen Pfad in einer Baumstruktur repräsentiert.

Offener Terminalpool
open terminal pool

Terminalpool, der nicht auf *Clients* eines Rechners oder eines bestimmten Typs beschränkt ist. An diesen Terminalpool können sich alle Clients anschließen, für die kein Rechner- oder Typ-spezifischer Terminalpool generiert ist.

OMNIS (BS2000-Systeme)
OMNIS

OMNIS ist ein „Session-Manager“ auf einem BS2000-System, der die gleichzeitige Verbindungsaufnahme von einem Terminal zu mehreren Partnern in einem Netzwerk ermöglicht. OMNIS ermöglicht es außerdem, mit *Multiplex-Verbindungen* zu arbeiten.

Online-Import
online import

Als Online-Import wird in einer *UTM-Cluster-Anwendung* das Importieren von Anwendungsdaten aus einer normal beendeten Knoten-Anwendung in eine laufende Knoten-Anwendung bezeichnet.

Online-Update
online update

Als Online-Update wird in einer *UTM-Cluster-Anwendung* die Änderung der Konfiguration der Anwendung oder des Anwendungsprogramms oder der Einsatz einer neuen UTM-Korrekturstufe bei laufender *UTM-Cluster-Anwendung* bezeichnet.

OpenCPIC

Trägersystem für UTM-Clients, die das *OSI TP* Protokoll verwenden.

OpenCPIC-Client
OpenCPIC client

OSI TP Partner-Anwendungen mit Trägersystem *OpenCPIC*.

openSM2

Die Produktlinie openSM2 ist eine einheitliche Lösung für das unternehmensweite Performance Management von Server- und Speichersystemen. openSM2 bietet eine Messdatenerfassung, Online-Überwachung und Offline-Auswertung.

openUTM-Cluster
openUTM cluster

aus der Sicht von UPIC-Clients, **nicht** aus Server-Sicht:
Zusammenfassung mehrerer Knoten-Anwendungen einer UTM-Cluster-Anwendung zu einer logischen Anwendung, die über einen gemeinsamen Symbolic Destination Name adressiert wird.

openUTM-D

openUTM-D (openUTM-Distributed) ist eine openUTM-Komponente, die *verteilte Verarbeitung* ermöglicht. openUTM-D ist integraler Bestandteil von openUTM.

OSI-LPAP-Bündel
OSI-LPAP bundle

LPAP-Bündel für *OSI TP*-Partner-Anwendungen.

OSI-LPAP-Partner
OSI-LPAP partner

OSI-LPAP-Partner sind die bei openUTM generierten Adressen der *OSI TP-Partner*. Für die *verteilte Verarbeitung* über das Protokoll *OSI TP* muss in der lokalen Anwendung für jede Partner-Anwendung ein OSI-LPAP-Partner konfiguriert werden. Der OSI-LPAP-Partner spiegelt in der lokalen Anwendung die Partner-Anwendung wider. Bei der Kommunikation wird die Partner-Anwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten OSI-LPAP-Partners angesprochen.

OSI-Referenzmodell
OSI reference model

Das OSI-Referenzmodell stellt einen Rahmen für die Standardisierung der Kommunikation von offenen Systemen dar. ISO, die Internationale Organisation für Standardisierung, hat dieses Modell im internationalen Standard
ISO IS7498 beschrieben. Das OSI-Referenzmodell unterteilt die für die Kommunikation von Systemen notwendigen Funktionen in sieben logische Schichten. Diese Schichten haben jeweils klar definierte Schnittstellen zu den benachbarten Schichten.

OSI TP

Von der ISO definiertes Kommunikationsprotokoll für die verteilte Transaktionsverarbeitung. OSI TP steht für Open System Interconnection Transaction Processing.

OSI TP-Partner
OSI TP partner

Partner der UTM-Anwendung, der mit der UTM-Anwendung über das OSI TP-Protokoll kommuniziert.
Beispiele für solche Partner sind:

- eine UTM-Anwendung, die über OSI TP kommuniziert
- eine Anwendung im IBM-Umfeld (z.B. CICS), die über openUTM-LU62 angeschlossen ist
- ein *OpenCPIC-Client*
- Anwendungen anderer TP-Monitore, die OSI TP unterstützen

Outbound-Conversation (CPI-C)
outbound conversation

siehe *Outgoing-Conversation*.

Outgoing-Conversation (CPI-C)
outgoing conversation

Eine Conversation, bei der das lokale CPI-C-Programm der *Initiator* ist, heißt Outgoing-Conversation. In der X/Open-Specification wird für Outgoing-Conversation auch das Synonym Outbound-Conversation verwendet.

Pagepool
page pool

Teil der *KDCFILE*, in dem Anwenderdaten gespeichert werden.
In einer *stand-alone Anwendung* sind dies z.B. *Dialog-Nachrichten*, Nachrichten an *Message Queues*, *Sekundärspeicherbereiche*.
In einer *UTM-Cluster-Anwendung* sind dies z.B. Nachrichten an *Message Queues, TLS*.

Parameterbereich
parameter area

Datenstruktur, in der ein *Teilprogramm* bei einem UTM-Aufruf die für diesen Aufruf notwendigen Operanden an openUTM übergibt.

Partner-Anwendung
partner application

Partner einer UTM-Anwendung bei *verteilter Verarbeitung*. Für die verteilte Verarbeitung werden höhere Kommunikationsprotokolle verwendet (*LU6.1*, *OSI TP* oder LU6.2 über das Gateway openUTM-LU62).

Postselection (BS2000-Systeme)
postselection

Auswahl der protokollierten UTM-Ereignisse aus der SAT-Protokolldatei, die ausgewertet werden sollen. Die Auswahl erfolgt mit Hilfe des Tools SATUT.

Programmraum (BS2000-Systeme)
program space

In Speicherklassen aufgeteilter virtueller Adressraum des BS2000, in dem sowohl ablauffähige Programme als auch reine Daten adressiert werden.

Prepare to commit (PTC)
prepare to commit

Bestimmter Zustand einer verteilten Transaktion:
Das Transaktionsende der verteilten Transaktion wurde eingeleitet, es wird jedoch noch auf die Bestätigung des Transaktionsendes durch den Partner gewartet.

Preselection (BS2000-Systeme)
preselection

Festlegung der für die *SAT-Beweissicherung* zu protokollierenden UTM-Ereignisse. Die Preselection erfolgt durch die UTM-SAT-Administration. Man unterscheidet Ereignis-spezifische, Benutzer-spezifische und Auftrags-(TAC-)spezifische Preselection.

Presentation-Selektor
presentation selector

Der Presentation-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Darstellungsschicht des *OSI-Referenzmodells*.

Primärspeicherbereich
primary storage area

Bereich im Arbeitsspeicher, auf den das *KDCS-Teilprogramm* direkt zugreifen kann, z.B. *Standard Primärer Arbeitsbereich*, *Kommunikationsbereich*.

Printerprozess (Unix- und Linux-Systeme)
printer process

siehe *Druckerprozess*.

Programmschnittstelle zur Administration
program interface for administration

UTM-Programmschnittstelle, mit deren Hilfe der Anwender eigene *Administrationsprogramme* erstellen kann. Die Programmschnittstelle zur Administration bietet u.a. Funktionen zur *dynamischen Konfiguration*, zur Modifikation von Eigenschaften und Anwendungsparametern und zur Abfrage von Informationen zur *Konfiguration* und zur aktuellen Auslastung der Anwendung.

Prozess
prozess

In den openUTM-Handbüchern wird der Begriff "Prozess" als Oberbegriff für Prozess (Unix-, Linux- und Windows-Systeme) und Task (BS2000-Systeme) verwendet.

Queue
queue

siehe *Message Queue*

Quick Start Kit

Beispielanwendung, die mit openUTM (Windows-Systeme) ausgeliefert wird.

Quittungs-Auftrag
confirmation job

Bestandteil eines *Auftrags-Komplexes*, worin der Quittungs-Auftrag dem *Basis-Auftrag* zugeordnet ist. Es gibt positive und negative Quittungsaufträge. Bei positivem Ergebnis des *Basis-Auftrags* wird der positive Quittungs-Auftrag wirksam, sonst der negative.

Redelivery
redelivery

Erneutes Zustellen einer *Asynchron-Nachricht*, nachdem diese nicht ordnungsgemäß verarbeitet werden konnte, z.B. weil die *Transaktion* zurückgesetzt oder der *Asynchron-Vorgang* abnormal beendet wurde. Die Nachricht wird wieder in die Message Queue eingereiht und lässt sich damit erneut lesen und/oder verarbeiten.

Reentrant-fähiges Programm
reentrant program

Programm, dessen Code durch die Ausführung nicht verändert wird.
Auf BS2000-Systemen ist dies Voraussetzung dafür, *Shared Code* zu nutzen.

Request
request

Anforderung einer *Service-Funktion* durch einen *Client* oder einen anderen Server.

Requestor
requestor

In XATMI steht der Begriff Requestor für eine Anwendung, die einen Service aufruft.

Resource Manager
resource manager

Resource Manager (RMs) verwalten Datenressourcen. Ein Beispiel für RMs sind Datenbank-Systeme. openUTM stellt aber auch selbst Resource Manager zur Verfügung, z.B. für den Zugriff auf *Message Queues*, lokale Speicherbereiche und Logging-Dateien. Anwendungsprogramme greifen auf RMs über RM-spezifische Schnittstellen zu. Für Datenbank-Systeme ist dies meist SQL, für die openUTM-RMs die Schnittstelle KDCS.

RFC1006

Von IETF (Internet Engineering Task Force) definiertes Protokoll der TCP/IP-Familie zur Realisierung der ISO-Transportdienste (Transportklasse 0) auf TCP/IP-Basis.

RSA

Abkürzung für die Erfinder des RSA-Verschlüsselungsverfahrens Rivest, Shamir und Adleman. Bei diesem Verfahren wird ein Schlüsselpaar verwendet, das aus einem öffentlichen und einem privaten Schlüssel besteht. Eine Nachricht wird mit dem öffentlichen Schlüssel verschlüsselt und kann nur mit dem privaten Schlüssel entschlüsselt werden. Das RSA-Schlüsselpaar wird von der UTM-Anwendung erzeugt.

SAT-Beweissicherung (BS2000-Systeme)
SAT audit

Beweissicherung durch die Komponente SAT (Security Audit Trail) des BS2000-Softwareproduktes SECOS.

SE Manager
SE manager

Web-basierte Benutzeroberfläche (GUI) für Business Server der SE Serie. Der SE Manager läuft auf der *Management Unit* und ermöglicht die zentrale Bedienung und Verwaltung von Server Units (mit /390-Architektur und/oder x86-Architektur), Application Units (x86-Architektur), Net Unit und der Peripherie.

SE Server
SE server

Ein Business Server der SE Serie von Fujitsu.

Sekundärspeicherbereich
secondary storage area

Transaktionsgesicherter Speicherbereich, auf den das KDCS-*Teilprogramm* mit speziellen Aufrufen zugreifen kann. Lokale Sekundärspeicherbereiche (LSSB) sind einem *Vorgang* zugeordnet, auf globale Sekundärspeicherbereiche (GSSB) kann von allen Vorgängen einer *UTM-Anwendung* zugegriffen werden. Weitere Sekundärspeicherbereiche sind der *Terminal-spezifische Langzeitspeicher (TLS)* und der *User-spezifische Langzeitspeicher (ULS)* .

Selektor
selector

Ein Selektor identifiziert im lokalen System einen *Zugriffspunkt* auf die Dienste einer Schicht des *OSI-Referenzmodells*. Jeder Selektor ist Bestandteil der Adresse des Zugriffspunktes.

Semaphor (Unix-, Linux- und Windows-Systeme)
semaphore

Betriebsmittel auf Unix-, Linux- und Windows-Systemen, das zur Steuerung und Synchronisation von Prozessen dient.

Server
server

Ein Server ist eine *Anwendung*, die *Services* zur Verfügung stellt. Oft bezeichnet man auch den Rechner, auf dem Anwendungen laufen, als Server.

Server-Seite einer Conversation (CPI-C)
server side of a conversation

Begriff ersetzt durch *Akzeptor*.

Server-Server-Kommunikation
server-server communication

siehe *verteilte Verarbeitung*.

Service Access Point

siehe *Dienstzugriffspunkt.*

Service
service

Services bearbeiten die Aufträge, die an eine Server-Anwendung geschickt werden. Ein Service in einer UTM-Anwendung wird auch Vorgang genannt und setzt sich aus einer oder mehreren Transaktionen zusammen. Ein Service wird über den Vorgangs-TAC aufgerufen. Services können von Clients oder anderen Services angefordert werden.

Service-gesteuerte Queue
service controlled queue

Message Queue, bei der der Abruf und die Weiterverarbeitung der Nachrichten durch Services gesteuert werden. Ein Service muss zum Lesen der Nachricht explizit einen KDCS-Aufruf (DGET) absetzen.
Service-gesteuerte Queues gibt es bei openUTM in den Varianten USER-Queue, TAC-Queue und Temporäre Queue.

Service Routine
service routine

siehe Teilprogramm.

Session
session

Kommunikationsbeziehung zweier adressierbarer Einheiten im Netz über das SNA-Protokoll LU6.1.

Session-Selektor
session selector

Der Session-Selektor identifiziert im lokalen System einen Zugriffspunkt zu den Diensten der Kommunikationssteuerschicht (Session-Layer) des OSI-Referenzmodells.

Shared Code (BS2000-Systeme)
shared code

Code, der von mehreren Prozessen gemeinsam benutzt werden kann.

Shared Memory
shared memory

Virtueller Speicherbereich, auf den mehrere Prozesse gleichzeitig zugreifen können.

Shared Objects (Unix-, Linux- und Windows-Systeme)
shared objects

Teile des Anwendungsprogramms können als Shared Objects erzeugt werden. Diese werden dynamisch zur Anwendung dazugebunden und können im laufenden Betrieb ausgetauscht werden. Shared Objects werden mit der KDCDEF-Anweisung SHARED-OBJECT definiert.

Sicherungspunkt
synchronization point, consistency point

Ende einer Transaktion. Zu diesem Zeitpunkt werden alle in der Transaktion vorgenommenen Änderungen der Anwendungsinformation gegen Systemausfall gesichert und für andere sichtbar gemacht. Während der Transaktion gesetzte Sperren werden wieder aufgehoben.

Single System Image

Unter single system image versteht man die Eigenschaft eines Clusters, nach außen hin als ein einziges, in sich geschlossenes System zu erscheinen. Die heterogene Natur des Clusters und die interne Verteilung der Ressourcen im Cluster ist für die Benutzer des Clusters und die Anwendungen, die mit dem Cluster kommunizieren, nicht sichtbar.

SOA

SOA (Service-oriented architecture).
SOA ist ein Konzept für eine Systemarchitektur, in dem Funktionen in Form von wieder verwendbaren, technisch voneinander unabhängigen und fachlich lose gekoppelten Services implementiert werden. Services können unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden, deren Spezifikationen öffentlich und damit vertrauenswürdig sein können. Service-Interaktion findet über eine dafür vorgesehene Kommunikationsinfrastruktur statt.

SOAP

SOAP (Simple Object Access Protocol) ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP stützt sich auf die Dienste anderer Standards, XML zur Repräsentation der Daten und Internet-Protokolle der Transport- und Anwendungsschicht zur Übertragung der Nachrichten.

Socket-Verbindung
socket connection

Transportsystem-Verbindung, die die Socket-Schnittstelle verwendet. Die Socket-Schnittstelle ist eine Standard-Programmschnittstelle für die Kommunikation über TCP/IP.

Stand-alone Anwendung
stand-alone application

siehe stand-alone UTM-Anwendung.

Stand-alone UTM-Anwendung
stand-alone UTM application

Herkömmliche UTM-Anwendung, die nicht Bestandteil einer UTM-Cluster-Anwendung ist.

Standard Primärer Arbeitsbereich/SPAB (KDCS)
standard primary working area

Bereich im Arbeitsspeicher, der jedem KDCS-Teilprogramm zur Verfügung steht. Sein Inhalt ist zu Beginn des Teilprogrammlaufs undefiniert oder mit einem Füllzeichen vorbelegt.

Startformat
start format

Format, das openUTM am Terminal ausgibt, wenn sich ein Benutzer erfolgreich bei der UTM-Anwendung angemeldet hat (ausgenommen nach Vorgangs-Wiederanlauf und beim Anmelden über Anmelde-Vorgang).

Statische Konfiguration
static configuration

Festlegen der Konfiguration bei der UTM-Generierung mit Hilfe des UTM-Tools KDCDEF.

SYSLOG-Datei
SYSLOG file

siehe System-Protokolldatei.

System-Protokolldatei
system log file

Datei oder Dateigeneration, in die openUTM während des Laufs einer UTM-Anwendung alle UTM-Meldungen protokolliert, für die das Meldungsziel SYSLOG definiert ist.

TAC
TAC

siehe Transaktionscode.

TAC-Queue
TAC queue

Message Queue, die explizit per KDCDEF-Anweisung generiert wird. Eine TAC-Queue ist eine Service-gesteuerte Queue und kann unter dem generierten Namen von jedem Service aus angesprochen werden.

Teilprogramm
program unit

UTM-Services werden durch ein oder mehrere Teilprogramme realisiert. Die Teilprogramme sind Bestandteile des Anwendungsprogramms. Abhängig vom verwendeten API müssen sie KDCS-, XATMI- oder CPIC-Aufrufe enthalten. Sie sind über Transaktionscodes ansprechbar. Einem Teilprogramm können mehrere Transaktionscodes zugeordnet werden.

Temporäre Queue
temporary queue

Message Queue, die dynamisch per Programm erzeugt wird und auch wieder per Programm gelöscht werden kann, vgl. Service-gesteuerte Queue.

Terminal-spezifischer Langzeitspeicher/TLS (KDCS)
terminal-specific long-term storage

Sekundärspeicher, der einem LTERM-, LPAP- oder OSI-LPAP-Partner zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

Timerprozess (Unix-, Linux- und Windows-Systeme)
timer process

Prozess, der Aufträge zur Zeitüberwachung von Workprozessen entgegennimmt, sie in ein Auftragsbuch einordnet und nach einer im Auftragsbuch festgelegten Zeit den Workprozessen zur Bearbeitung wieder zustellt.

TLS Termination Proxy
TLS termination proxy

Ein TLS-Terminierungsproxy ist ein Proxy-Server, der verwendet wird, um eingehende TLS-Verbindungen zu verarbeiten, die Daten zu entschlüsseln und die unverschlüsselte Anforderung an andere Server weiterzugeben.

TNS (Unix-, Linux- und Windows-Systeme)

Abkürzung für den Transport Name Service, der einem Anwendungsnamen einen Transport-Selektor und das Transportsystem zuordnet, über das die Anwendung erreichbar ist.

Tomcat

siehe Apache Tomcat

Transaktion
transaction

Verarbeitungsabschnitt innerhalb eines Services, für den die Einhaltung der ACID-Eigenschaften garantiert wird. Von den in einer Transaktion beabsichtigten Änderungen der Anwendungsinformation werden entweder alle konsistent durchgeführt oder es wird keine durchgeführt (Alles-oder-Nichts Regel). Das Transaktionsende bildet einen Sicherungspunkt.

Transaktionscode/TAC
transaction code

Name, über den ein Teilprogramm aufgerufen werden kann. Der Transaktionscode wird dem Teilprogramm bei der statischen oder dynamischen Konfiguration zugeordnet. Einem Teilprogramm können auch mehrere Transaktionscodes zugeordnet werden.

Transaktionsrate
transaction rate

Anzahl der erfolgreich beendeten Transaktionen pro Zeiteinheit.

Transfer-Syntax
transfer syntax

Bei OSI TP werden die Daten zur Übertragung zwischen zwei Rechnersystemen von der lokalen Darstellung in die Transfer-Syntax umgewandelt. Die Transfer-Syntax beschreibt die Daten in einem neutralen Format, das von allen beteiligten Partnern verstanden wird. Jeder Transfer-Syntax muss ein Object Identifier zugeordnet sein.

Transport Layer Security
transport layer security

Der Transport Layer Security, ist ein hybrides Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet.

Transport-Selektor
transport selector

Der Transport-Selektor identifiziert im lokalen System einen Dienstzugriffspunkt zur Transportschicht des OSI-Referenzmodells.

Transportsystem-Anwendung
transport system application

Anwendung, die direkt auf einer Transportsystem-Schnittstelle wie z.B. CMX, DCAM oder Socket aufsetzt. Für den Anschluss von Transportsystem-Anwendungen muss bei der Konfiguration als Partnertyp APPLI oder SOCKET angegeben werden. Eine Transportsystem-Anwendung kann nicht in eine Verteilte Transaktion eingebunden werden.

Transportsystem-Endpunkt
transport system end point

Bei der Client-/Server- oder Server-/Server-Kommunikation wird eine Verbindung zwischen zwei Transportsystem-Endpunkten aufgebaut. Ein Transportsystem-Endpunkt wird auch als lokaler Anwendungsname bezeichnet und wird mit der Anweisung BCAMAPPL oder mit MAX APPLINAME definiert.

Transportsystem-Zugriffspunkt
transport system access point

siehe Transportsystem-Endpunkt.

Transportverbindung
transport connection

Im OSI-Referenzmodell eine Verbindung zwischen zwei Instanzen der Schicht 4 (Transportschicht).

TS-Anwendung
TS application

siehe Transportsystem-Anwendung.

Typisierter Puffer (XATMI)
typed buffer

Puffer für den Austausch von typisierten und strukturierten Daten zwischen Kommunikationspartnern. Durch diese typisierten Puffer ist die Struktur der ausgetauschten Daten den Partnern implizit bekannt.

UPIC

Trägersystem für UTM-Clients. UPIC steht für Universal Programming Interface for Communication. Die Kommunikation mit der UTM-Anwendung erfolgt über das UPIC-Protokoll.

UPIC-Client

Bezeichnung für UTM-Clients mit Trägersystem UPIC und JConnect-Clients.

UPIC-Protokoll
Upic protocol

Protokoll für die Client-Server-Kommunikation mit UTM-Anwendungen. Das UPIC-Protokoll wird von UPIC-Clients und von JConnect-Clients verwendet.

UPIC Analyzer

Komponente zur Analyse der mit UPIC Capture mitgeschnittenen UPIC-Kommunikation. Dieser Schritt dient dazu, den Mitschnitt für das Abspielen mit UPIC Replay aufzubereiten.

UPIC Capture

Mitschneiden der Kommunikation zwischen UPIC-Clients und UTM-Anwendungen, um sie zu einem späteren Zeitpunkt abspielen zu können
(UPIC Replay).

UPIC Replay

Komponente zum Abspielen der mit UPIC Capture mitgeschnittenen und mit UPIC Analyzer aufbereiteten UPIC-Kommunikation.

USER-Queue
USER queue

Message Queue, die openUTM jeder Benutzerkennung zur Verfügung stellt. Eine USER-Queue zählt zu den Service-gesteuerten Queues und ist immer der jeweiligen Benutzerkennung zugeordnet. Der Zugriff von fremden UTM-Benutzern auf die eigene USER-Queue kann eingeschränkt werden.

User-spezifischer Langzeitspeicher/ULS
user-specific long-term storage

Sekundärspeicher, der einer Benutzerkennung, einer Session oder einer Association zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

USLOG-Datei
USLOG file

siehe Benutzer-Protokolldatei.

UTM-Anwendung
UTM application

Eine UTM-Anwendung stellt Services zur Verfügung, die Aufträge von Clients oder anderen Anwendungen bearbeiten. openUTM übernimmt dabei u.a. die Transaktionssicherung und das Management der Kommunikations- und Systemressourcen. Technisch gesehen ist eine UTM-Anwendung eine Prozessgruppe, die zur Laufzeit eine logische Server-Einheit bildet.

UTM-Client
UTM client

siehe Client.

UTM-Cluster-Anwendung
UTM cluster application

UTM-Anwendung, die für den Einsatz in einem Cluster generiert ist und die man logisch als **eine** Anwendung betrachten kann.
Physikalisch gesehen besteht eine UTM-Cluster-Anwendung aus mehreren, identisch generierten UTM-Anwendungen, die auf den einzelnen Knoten laufen.

UTM-Cluster-Dateien
UTM cluster files

Oberbegriff für alle Dateien, die für den Ablauf einer UTM-Cluster-Anwendung auf Unix-, Linux- und Windows-Systemen benötigt werden. Dazu gehören folgende Dateien:

- Cluster-Konfigurationsdatei
- Cluster-User-Datei
- Dateien des Cluster-Pagepool
- Cluster-GSSB-Datei
- Cluster-ULS-Datei
- Dateien des Cluster-Administrations-Journals*
- Cluster-Lock-Datei*
- Lock-Datei zur Start-Serialisierung*

Die mit * gekennzeichneten Dateien werden beim Start der ersten Knoten-Anwendung angelegt, alle anderen Dateien werden bei der Generierung mit KDCDEF erzeugt.

UTM-D

siehe openUTM-D.

UTM-Datenstation
UTM terminal

Begriff ersetzt durch LTERM-Partner.

UTM-F

UTM-Anwendungen können als UTM-F-Anwendungen (UTM-Fast) generiert werden. Bei UTM-F wird zugunsten der Performance auf Platteneingaben/-ausgaben verzichtet, mit denen bei UTM-S die Sicherung von Benutzer- und Transaktionsdaten durchgeführt wird. Gesichert werden lediglich Änderungen der Verwaltungsdaten.
In UTM-Cluster-Anwendungen, die als UTM-F-Anwendung generiert sind (APPLIMODE=FAST), werden Cluster-weit gültige Anwenderdaten auch gesichert. Dabei werden GSSB- und ULS-Daten genauso behandelt wie in UTM-Cluster-Anwendungen, die mit UTM-S generiert sind. Vorgangs-Daten von Benutzern mit RESTART=YES werden jedoch nur beim Abmelden des Benutzers anstatt bei jedem Transaktionsende geschrieben.

UTM-Generierung
UTM generation

Statische Konfiguration einer UTM-Anwendung mit dem UTM-Tool KDCDEF und Erzeugen des Anwendungsprogramms.

UTM-gesteuerte Queues
UTM controlled queue

Message Queues, bei denen der Abruf und die Weiterverarbeitung der Nachrichten vollständig durch openUTM gesteuert werden. Siehe auch Asynchron-Auftrag, Hintergrund-Auftrag und Asynchron-Nachricht.

UTM-S

Bei UTM-S-Anwendungen sichert openUTM neben den Verwaltungsdaten auch alle Benutzerdaten über ein Anwendungsende und einen Systemausfall hinaus. Außerdem garantiert UTM-S bei allen Störungen die Sicherheit und Konsistenz der Anwendungsdaten. Im Standardfall werden UTM-Anwendungen als UTM-S-Anwendungen (UTM-Secure) generiert.

UTM-SAT-Administration (BS2000-Systeme)
UTM SAT administration

Durch die UTM-SAT-Administration wird gesteuert, welche sicherheitsrelevanten UTM-Ereignisse, die im Betrieb der UTM-Anwendung auftreten, von SAT protokolliert werden sollen. Für die UTM-SAT-Administration wird eine besondere Berechtigung benötigt.

UTM-Seite
UTM page

Ist eine Speichereinheit, die entweder 2K, 4K oder 8K umfasst. In stand-alone UTM-Anwendungen kann die Größe einer UTM-Seite bei der Generierung der UTM-Anwendung auf 2K, 4K oder 8K gesetzt werden. In einer UTM-Cluster-Anwendung ist die Größe einer UTM-Seite immer 4K oder 8K. Pagepool und Wiederanlauf-Bereich der KDCFILE sowie UTM-Cluster-Dateien werden in Einheiten der Größe einer UTM-Seite unterteilt.

UTM Socket Protokoll (USP)
UTM socket protocol

Proprietäres Protokoll von openUTM oberhalb von TCP/IP zur Umsetzung der über die Socket-Schnittstelle empfangenen Bytestreams in Nachrichten.

UTM-System-Prozess
UTM system process

UTM-Prozess, der zusätzlich zu den per Startparameter angegebenen Prozessen gestartet wird und nur ausgewählte Aufträge bearbeitet. UTM-System-Prozesse dienen dazu, eine UTM-Anwendung auch bei sehr hoher Last reaktionsfähig zu halten.

UTM-Tool
UTM tool

Programm, das zusammen mit openUTM zur Verfügung gestellt und für bestimmte UTM-spezifische Aufgaben benötigt wird (z.B. zum Konfigurieren).

utmpfad (Unix-, Linux- und Windows-Systeme)
utmpath

Das Dateiverzeichnis unter dem die Komponenten von openUTM installiert sind, wird in diesem Handbuch als utmpfad bezeichnet.
Um einen korrekten Ablauf von openUTM zu garantieren, muss die Umgebungsvariable UTMPATH auf den Wert von utmpfad gesetzt werden. Auf Unix- und Linux-Systemen müssen Sie UTMPATH vor dem Starten einer UTM-Anwendung setzen. Auf Windows-Systemen wird UTMPATH passend zu der zuletzt installierten UTM-Version gesetzt.

Verarbeitungsschritt
processing step

Ein Verarbeitungsschritt beginnt mit dem Empfangen einer Dialog-Nachricht, die von einem Client oder einer anderen Server-Anwendung an die UTM-Anwendung gesendet wird. Der Verarbeitungsschritt endet entweder mit dem Senden einer Antwort und beendet damit auch den Dialog-Schritt oder er endet mit dem Senden einer Dialog-Nachricht an einen Dritten.

Verbindungs-Benutzerkennung
connection user ID

Benutzerkennung, unter der eine TS-Anwendung oder ein UPIC-Client direkt nach dem Verbindungsaufbau bei der UTM-Anwendung angemeldet wird. Abhängig von der Generierung des Clients (= LTERM-Partner) gilt:

- Die Verbindungs-Benutzerkennung ist gleich dem USER der LTERM-Anweisung (explizite Verbindungs-Benutzerkennung). Eine explizite Verbindungs-Benutzerkennung muss mit einer USER-Anweisung generiert sein und kann nicht als "echte" Benutzerkennung verwendet werden.
- Die Verbindungs-Benutzerkennung ist gleich dem LTERM-Partner (implizite Verbindungs-Benutzerkennung), wenn bei der LTERM-Anweisung kein USER angegeben wurde oder wenn ein LTERM-Pool generiert wurde.

In einer UTM-Cluster-Anwendung ist der Vorgang einer Verbindungs-Benutzerkennung (RESTART=YES bei LTERM oder USER) an die Verbindung gebunden und damit Knoten-lokal. Eine Verbindungs-Benutzerkennung, die mit RESTART=YES generiert ist, kann in jeder Knoten-Anwendung einen eigenen Vorgang haben.

Verbindungsbündel
connection bundle

siehe LTERM-Bündel.

Verschlüsselungsstufe
encryption level

Die Verschlüsselungsstufe legt fest, ob und inwieweit ein Client Nachrichten und Passwort verschlüsseln muss.

Verteilte Transaktion
distributed transaction

Transaktion, die sich über mehr als eine Anwendung erstreckt und in mehreren (Teil-)Transaktionen in verteilten Systemen ausgeführt wird.

Verteilte Transaktionsverarbeitung
Distributed Transaction Processing

Verteilte Verarbeitung mit verteilten Transaktionen.

Verteilte Verarbeitung
distributed processing

Bearbeitung von Dialog-Aufträgen durch mehrere Anwendungen oder Übermittlung von Hintergrundaufträgen an eine andere Anwendung. Für die verteilte Verarbeitung werden die höheren Kommunikationsprotokolle LU6.1 und OSI TP verwendet. Über openUTM-LU62 ist verteilte Verarbeitung auch mit LU6.2 Partnern möglich. Man unterscheidet verteilte Verarbeitung mit verteilten Transaktionen (Anwendungs-übergreifende Transaktionssicherung) und verteilte Verarbeitung ohne verteilte Transaktionen (nur lokale Transaktionssicherung). Die verteilte Verarbeitung wird auch Server-Server-Kommunikation genannt.

Vorgang (KDCS)
service

Ein Vorgang dient zur Bearbeitung eines Auftrags in einer UTM-Anwendung. Er setzt sich aus einer oder mehreren Transaktionen zusammen. Die erste Transaktion wird über den Vorgangs-TAC aufgerufen. Es gibt Dialog-Vorgänge und Asynchron-Vorgänge. openUTM stellt den Teilprogrammen eines Vorgangs gemeinsame Datenbereiche zur Verfügung. Anstelle des Begriffs Vorgang wird häufig auch der allgemeinere Begriff Service gebraucht.

Vorgangs-Kellerung (KDCS)
service stacking

Ein Terminal-Benutzer kann einen laufenden Dialog-Vorgang unterbrechen und einen neuen Dialog-Vorgang einschieben. Nach Beendigung des eingeschobenen Vorgangs wird der unterbrochene Vorgang fortgesetzt.

Vorgangs-Kettung (KDCS)
service chaining

Bei Vorgangs-Kettung wird nach Beendigung eines Dialog-Vorgangs ohne Angabe einer Dialog-Nachricht ein Folgevorgang gestartet.

Vorgangs-TAC (KDCS)
service TAC

Transaktionscode, mit dem ein Vorgang gestartet wird.

Vorgangs-Wiederanlauf (KDCS)
service restart

Wird ein Vorgang unterbrochen, z.B. infolge Abmeldens des Terminal-Benutzers oder Beendigung der UTM-Anwendung, führt openUTM einen Vorgangs-Wiederanlauf durch. Ein Asynchron-Vorgang wird neu gestartet oder beim zuletzt erreichten Sicherungspunkt fortgesetzt, ein Dialog-Vorgang wird beim zuletzt erreichten Sicherungspunkt fortgesetzt. Für den Terminal-Benutzer wird der Vorgangs-Wiederanlauf eines Dialog-Vorgangs als Bildschirm-Wiederanlauf sichtbar, sofern am letzten Sicherungspunkt eine Dialog-Nachricht an den Terminal-Benutzer gesendet wurde.

Warmstart
warm start

Start einer UTM-S-Anwendung nach einer vorhergehenden abnormalen Beendigung. Dabei wird die Anwendungsinformation auf den zuletzt erreichten konsistenten Zustand gesetzt. Unterbrochene Dialog-Vorgänge werden dabei auf den zuletzt erreichten Sicherungspunkt zurückgesetzt, so dass die Verarbeitung an dieser Stelle wieder konsistent aufgenommen werden kann (Vorgangs-Wiederanlauf). Unterbrochene Asynchron-Vorgänge werden zurückgesetzt und neu gestartet oder beim zuletzt erreichten Sicherungspunkt fortgesetzt.
Bei UTM-F-Anwendungen werden beim Start nach einer vorhergehenden abnormalen Beendigung lediglich die dynamisch geänderten Konfigurationsdaten auf den zuletzt erreichten konsistenten Zustand gesetzt.
In UTM-Cluster-Anwendungen werden die globalen Sperren auf GSSB und ULS, die bei der abnormalen Beendigung von dieser Knoten-Anwendung gehalten wurden, aufgehoben. Außerdem werden Benutzer, die zum Zeitpunkt der abnormalen Beendigung an dieser Knoten-Anwendung angemeldet waren, abgemeldet.

Web Service
web service

Anwendung, die auf einem Web-Server läuft und über eine standardisierte und programmatische Schnittstelle (öffentlich) verfügbar ist. Die Web Services-Technologie ermöglicht es, UTM-Teilprogramme für moderne Web-Client-Anwendungen verfügbar zu machen, unabhängig davon, in welcher Programmiersprache sie entwickelt wurden.

WebAdmin
WebAdmin

Web-basiertes Tool zur Administration von openUTM-Anwendungen über Web-Browser. WebAdmin enthält neben dem kompletten Funktionsumfang der Programmschnittstelle zur Administration noch zusätzliche Funktionen.

Wiederanlauf
restart

siehe Bildschirm-Wiederanlauf,
siehe Vorgangs-Wiederanlauf.

WinAdmin
WinAdmin

Java-basiertes Tool zur Administration von openUTM-Anwendungen über eine grafische Oberfläche. WinAdmin enthält neben dem kompletten Funktionsumfang der Programmschnittstelle zur Administration noch zusätzliche Funktionen.

Workload Capture & Replay
workload capture & replay

Programmfamilie zur Simulation von Lastsituationen, bestehend aus den Haupt-Komponenten UPIC Capture, UPIC Analyzer und Upic Replay und auf Unix-, Linux- und Windows-Systemen dem Dienstprogramm kdcsort. Mit Workload Capture & Replay lassen sich UPIC-Sessions mit UTM-Anwendungen aufzeichnen, analysieren und mit veränderten Lastparametern wieder abspielen.

Workprozess (Unix-, Linux- und Windows-Systeme)
work process

Prozess, in dem die Services der UTM-Anwendung ablaufen.

WS4UTM

WS4UTM (**W**eb**S**ervices for open**UTM**) ermöglicht es Ihnen, auf komfortable Weise einen Service einer UTM-Anwendung als Web Service zur Verfügung zu stellen.

XATMI

XATMI (X/Open Application Transaction Manager Interface) ist eine von X/Open standardisierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen.
Das in openUTM implementierte XATMI genügt der XATMI CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. XATMI in openUTM kann über die Protokolle OSI TP, LU6.1 und UPIC kommunizieren.

XHCS (BS2000-Systeme)

XHCS (Extended Host Code Support) ist ein BS2000-Softwareprodukt für die Unterstützung internationaler Zeichensätze.

XML

XML (eXtensible Markup Language) ist eine vom W3C (WWW-Konsortium) genormte Metasprache, in der Austauschformate für Daten und zugehörige Informationen definiert werden können.

Zeitgesteuerter Auftrag
time-driven job

Auftrag, der von openUTM bis zu einem definierten Zeitpunkt in einer Message Queue zwischengespeichert und dann an den Empfänger weitergeleitet wird. Empfänger kann sein: ein Asynchron-Vorgang der selben Anwendung, eine TAC-Queue, eine Partner-Anwendung, ein Terminal oder ein Drucker. Zeitgesteuerte Aufträge können nur von KDCS-Teilprogrammen erteilt werden.

Zugangskontrolle
system access control

Prüfung durch openUTM, ob eine bestimmte Benutzerkennung berechtigt ist, mit der UTM-Anwendung zu arbeiten. Die Berechtigungsprüfung entfällt, wenn die UTM-Anwendung ohne Benutzerkennungen generiert wurde.

Zugriffskontrolle
data access control

Prüfung durch openUTM, ob der Kommunikationspartner berechtigt ist, auf ein bestimmtes Objekt der Anwendung zuzugreifen. Die Zugriffsrechte werden als Bestandteil der Konfiguration festgelegt.

Zugriffspunkt
access point

siehe Dienstzugriffspunkt.

14 Abkürzungen

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-Lader-Starter (BS2000-Systeme)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Codierter Zeichensatz (Coded Character Set)
CCSN	Name des codierten Zeichensatzes (Coded Character Set Name)
CICS	Customer Information Control System (IBM)
CID	Control Identification
CMX	Communication Manager in Unix-, Linux- und Windows-Systemen
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000-Systeme)
DB	Database
DBH	Database Handler
DC	Data Communication
DCAM	Data Communication Access Method
DES	Data Encryption Standard

DLS	Distributed Lock Manager (BS2000-Systeme)
DMS	Data Management System
DNS	Domain Name Service
DSS	Datensichtstation (=Terminal)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
DVS	Datenverwaltungssystem
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	Globaler Sekundärer Speicherbereich
HIPLEX®	Highly Integrated System Complex (BS2000-Systeme)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IFG	Interaktiver Format-Generator
ILCS	Inter Language Communication Services (BS2000-Systeme)
IMS	Information Management System (IBM)
IPC	Inter-Process-Communication
IRV	Internationale Referenzversion
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	Kommunikationsbereich

KBPROG	KB-Programmbereich
KDCADMI	KDC Administration Interface
KDCS	Kompatible Datenkommunikationsschnittstelle
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000-Systeme)
LSSB	Lokaler Sekundärer Speicherbereich
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000-Systeme)
NB	Nachrichtenbereich
NEA	Netzwerkarchitektur bei BS2000-Systemen
NFS	Network File System/Service
NLS	Unterstützung der Landessprache (Native Language Support)
OLTP	Online Transaction Processing
OML	Object Modul Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Prozess-Identifikation
PIN	Persönliche Identifikationsnummer
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Rechenzentrums-Abrechnungs-Verfahren
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption-Algorithmus nach Rivest, Shamir, Adleman

RSO	Remote SPOOL Output (BS2000-Systeme)
RTS	Runtime System (Laufzeitsystem)
SAT	Security Audit Trail (BS2000-Systeme)
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 Primärer Arbeitsbereich
SQL	Structured Query Language
SSB	Sekundärer Speicherbereich
SSL	Secure Socket Layer
SSO	Single-Sign-On
TAC	Transaktionscode
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-spezifischer Langzeitspeicher
TLS	Transport Layer Security
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaktions-Betrieb)
TPR	Task privileged (privilegierter Funktionszustand des BS2000-Systems)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Task user (nicht privilegierter Funktionszustand des BS2000-Systems)

TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universelles Datenbanksystem
UDT	Unstructured Data Transfer
ULS	User-spezifischer Langzeitspeicher
UPIC	Universal Programming Interface for Communication
USP	UTM-Socket-Protokoll
UTM	Universeller Transaktionsmonitor
UTM-D	UTM-Funktionen für verteilte Verarbeitung („Distributed“)
UTM-F	Schnelle UTM-Variante („Fast“)
UTM-S	UTM-Sicherheitsvariante
UTM-XML	XML-Schnittstelle von openUTM
VGID	Vorgangs-Identifikation
VTSU	Virtual Terminal Support
VTV	Verteilte Transaktionsverarbeitung
VV	Verteilte Verarbeitung
WAN	Wide Area Network
WS4UTM	WebServices for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (Schnittstelle von X/Open zum Zugriff auf 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

15 Literatur

Die Handbücher finden Sie im Internet unter https://bs2manuals.ts.fujitsu.com.

Dokumentation zu openUTM

openUTM
Konzepte und Funktionen
Benutzerhandbuch

openUTM
Anwendungen programmieren mit KDCS für COBOL, C und C++
Basishandbuch

openUTM
Anwendungen generieren
Benutzerhandbuch

openUTM
Einsatz von UTM-Anwendungen auf BS2000-Systemen
Benutzerhandbuch

openUTM
Einsatz von UTM-Anwendungen auf Unix-, Linux- und Windows-Systemen
Benutzerhandbuch

openUTM
Anwendungen administrieren
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose auf BS2000-Systemen
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose auf Unix-, Linux- und Windows-Systemen
Benutzerhandbuch

openUTM
Anwendungen erstellen mit X/Open-Schnittstellen
Benutzerhandbuch

openUTM
XML für openUTM

openUTM-Client (Unix-Systeme) **für Trägersystem OpenCPIC**
Client-Server-Kommunikation mit openUTM
Benutzerhandbuch

openUTM-Client für Trägersystem UPIC
Client-Server-Kommunikation mit openUTM
Benutzerhandbuch

openUTM WinAdmin
Grafischer Administrationsarbeitsplatz für openUTM
Beschreibung und Online-Hilfe

openUTM WebAdmin
Web-Oberfläche zur Administration von openUTM
Beschreibung und Online-Hilfe

openUTM, openUTM-LU62
Verteilte Transaktionsverarbeitung
zwischen openUTM und CICS-, IMS- und LU6.2-Anwendungen
Benutzerhandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Assembler
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Fortran
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Pascal-XT
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für PL/I
Ergänzung zum Basishandbuch

WS4UTM (Unix- und Windows-Systeme**)**
Web-Services für openUTM

Dokumentation zum openSEAS-Produktumfeld

BeanConnect
Benutzerhandbuch

openUTM-JConnect
Verbindung von Java-Clients zu openUTM
Benutzerdokumentation und Java-Docs

WebTransactions
Konzepte und Funktionen

WebTransactions
Template-Sprache

WebTransactions
Anschluss an openUTM-Anwendungen über UPIC

WebTransactions
Anschluss an MVS-Anwendungen

WebTransactions
Anschluss an OSD-Anwendungen

Dokumentation zum BS2000-Umfeld

AID Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

AID Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

AID Advanced Interactive Debugger
Testen von C/C++-Programmen
Benutzerhandbuch

BCAM
BCAM Band 1/2
Benutzerhandbuch

BINDER
Benutzerhandbuch

BS2000 OSD/BC
Kommandos Band 1-7
Benutzerhandbuch

BS2000 OSD/BC
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

BS2IDE
Eclipse-based Integrated Development Environment for BS2000
User Guide and Installation Guide
Webseite: *https://bs2000.ts.fujitsu.com/bs2ide/*

BLSSERV
Bindelader-Starter in BS2000/OSD
Benutzerhandbuch

DCAM
COBOL-Aufrufe
Benutzerhandbuch

DCAM
Makroaufrufe
Benutzerhandbuch

DCAM
Programmschnittstellen
Beschreibung

FHS
Formatierungssystem für openUTM, TIAM, DCAM
Benutzerhandbuch

IFG für FHS
Benutzerhandbuch

HIPLEX AF
Hochverfügbarkeit von Anwendungen in BS2000/OSD
Produkthandbuch

HIPLEX MSCF
BS2000-Rechner im Verbund
Benutzerhandbuch

IMON
Installationsmonitor
Benutzerhandbuch

LMS
SDF-Format
Benutzerhandbuch

MT9750 (MS Windows)
9750-Emulation unter Windows
Produkthandbuch

OMNIS/OMNIS-MENU
Funktionen und Kommandos
Benutzerhandbuch

OMNIS/OMNIS-MENU
Administration und Programmierung
Benutzerhandbuch

OSS (BS2000)
OSI Session Service
User Guide

openSM2
Software Monitor
Benutzerhandbuch

RSO
Remote SPOOL Output
Benutzerhandbuch

SECOS
Security Control System
Benutzerhandbuch

SECOS
Security Control System
Tabellenheft

SESAM/SQL
Datenbankbetrieb
Benutzerhandbuch

TIAM
Benutzerhandbuch

UDS/SQL
Datenbankbetrieb
Benutzerhandbuch

Unicode im BS2000/OSD
Übersichtshandbuch

VTSU
Virtual Terminal Support
Benutzerhandbuch

XHCS
8-bit-Code- und Unicode-Unterstützung im BS2000/OSD
Benutzerhandbuch

Dokumentation zum Umfeld von Unix-, Linux- und Windows-Systemen

CMX V6.0 (Unix-Systeme)
Betrieb und Administration
Benutzerhandbuch

CMX V6.0
CMX-Anwendungen programmieren
Programmierhandbuch

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTER™
Konzept (Solaris, Linux)
Benutzerhandbuch

openSM2
Die Dokumentation zu openSM2 wird in Form von ausführlichen Online-Hilfen bereitgestellt, die mit dem Produkt ausgeliefert werden.

Sonstige Literatur

CPI-C
X/Open CAE Specification
Distributed Transaction Processing:
The CPI-C Specification, Version 2
ISBN 1 85912 135 7

Reference Model
X/Open Guide
Distributed Transaction Processing:
Reference Model, Version 2
ISBN 1 85912 019 9

REST
Architectural Styles and the Design of Network-based Software Architectures
Dissertation Roy Fielding

TX
X/Open CAE Specification
Distributed Transaction Processing:
The TX (Transaction Demarcation) Specification
ISBN 1 85912 094 6

XATMI
X/Open CAE Secification
Distributed Transaction Processing
The XATMI Specification
ISBN 1 85912 130 6

XML
Spezifikation des W3C (www – Konsortium)
Webseite: *http://www.w3.org/XML*