Check Processing Control System
International MVS/ESA™

Programming Reference

Release 1

SC31-3997-03

Check Processing Control System
International MVS/ESA™

Programming Reference

Release 1

SC31-3997-03

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

| **Fourth Edition (November, 1999)**

This edition applies to Release 1 Modification 0 of the IBM Check Processing Control System International MVS/ESA (CPCS-I) licensed program (Program No. 5799-FKT). This publication is current as of PTF number UW61623.

Information in this manual is subject to change from time to time. Before using this publication in connection with the operation of IBM systems, consult your IBM representative to be sure you have the latest edition and any Technical Newsletters.

IBM does not stock publications at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader’s comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department 58G, MG96/204, 8501 IBM Drive, Charlotte, NC 28257-8563, U.S.A. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Notices xiii
Year 2000 Readiness xiv
Trademarks xiv

About This Publication xv
Who Should Read This Publication? xv
How Is This Publication Organized? xv
Related Publications xv
| Summary of Changes for SC31-3997-03. xvii

Chapter 1. CPCS-I Application Programming 1-1
Overview 1-3
Adding CPCS-I User Tasks 1-3
Accessing the Start Parameters 1-4
Returning to CPCS-I 1-7
Performing Terminal I/O 1-11
Determining Route Codes 1-21
Sending Messages to User Executive Tasks 1-27
Writing User Executive Tasks 1-33
Issuing Messages 1-39
Performing String I/O 1-47
Processing User Areas with the User Area Manager 1-57
Deleting Strings 1-60
Accessing/Updating Pass-to-Pass Control Information 1-64
Using the Tracer Group Update Utility 1-70
Accessing/Updating Cycle Control Information 1-77
Accessing Bank Control Information 1-81
Finding the Next Available Kill-Bundle Record 1-84
Obtaining Unique Sequence Numbers 1-86
Generating Spooled Reports 1-90
Generating JES Reports 1-97
Generating Columnar Reports 1-104
Generating Expanded Codeline Record Reports 1-113
Performing Security Checks 1-120
Creating DFTI Input Data Sets 1-123
Interfacing with Host Codeline Edit Routines 1-129
Invoking User Exits With the User Exit Manager 1-133

Chapter 2. CPCS-I User Exit Programming 2-1
Overview 2-3
| AllO Exit — Perform Document Processor Allocation/Unallocation Request
| Additional Validations 2-4
CDIF Exit — Modify DEFT Input Data Set Creation 2-7
CSBU Exits 1/2 — Modify Edit Routine and Table Load Processing 2-11
CYCL Exit — Validate Cycle and Endorse Dates 2-12
DFTI Exit 1 — Validate DEFT Input Initialization 2-15
DFTI Exit 2 — Modify DEFT Input Processing 2-19
DFTO Exit — Modify DEFT Output Processing 2-23
DFTP Exit — Analyze DFTP Activity 2-29
FSCN Exit — Modify FSCN Processing 2-33

HCDM Exit 1 — Modify HCDM Profile Parameters 2-37
HCDM Exit 2 — Modify HCDM Reconciliation String Creation 2-39
HCDM Exit 3 — Modify HCDM Unmatched Item Processing 2-44
HCDM Exit 4 — Modify HCDM Positional Match Sort Criteria 2-46
IDCR Exit — Access Adjustment Descriptions 2-48
MDIS Exit — Modify M-string Distribution 2-50
MICR Exit 1 — Customize MICR Entry Begin Parameter Validations 2-54
MICR Exit 2 — Customize MICR Document Processing 2-59
MICR Exit 3 — Customize MICR Codeline Data Match Processing 2-69
MICR Exit 4 — Customize Document Processor's Edit Routine and Table Load Processing (Non-XF Sort Types) 2-77
MICR Exits 5/6 — Customize Document Processor's Edit Routine and Table Load Processing (XF Sort Types) 2-83
MOLRI Exit — Modify Host's Edit Routine and Table Load Process 2-88
MRGE Exit — Modify HSRR MRGE Processing 2-93
MTASK Exit 1 — Perform Additional CPCS-I Startup Processing 2-95
MTASK Exit 2 — Perform Additional CPCS-I Shutdown Processing 2-97
ODCR Exit — Access Adjustment Descriptions 2-99
RCVY Exit — Modify Recovery Processing 2-100
SECR Exit — Perform User Security Validations 2-102
VSMGR Exit 1 — Perform CPCS-I Startup VSAM Data Set Processing . . 2-107
VSMGR Exit 2 — Perform CPCS-I Shutdown VSAM Data Set Processing 2-110
VTASK Exit 1 — VSCRL Special Buffer Processing 2-112
VTASK Exit 2 — VSCRL Special Scroll Initialization Processing 2-113
VTASK Exits 3/4 — VCOMS Special PCB Release Processing 2-114
VTASK Exit 5 — VEXTS Special Terminal Logon Processing 2-115

Chapter 3. Assembler Macros Provided by CPCS-I 3-1
Overview 3-3
ALDSN — Perform Dynamic Allocation Functions 3-4
ALLOC — Perform Dynamic Allocation Functions 3-10
CPCSNAME — Validate Name 3-16
DKNAMODE — Test/Set Addressing Mode 3-18
DKNFILL — Fill Segment with Padding Byte 3-21
DKNREGS — Generate Register Equates 3-23
DKNSBT — Search SCI Sort Binary Table 3-24
DKNXMODE — Test/Set Address Space Control Mode 3-26
FFLDL — Generate Field Length Equates/Constants 3-28
FPACK — Pack Unsigned a Zoned Numeric Field 3-31
FUNPK — Unpack a Packed Unsigned Field 3-34
XRETURN — Perform Program Exit Linkage 3-37

Chapter 4. Subroutines Provided by CPCS-I 4-1
Overview 4-3
DKNDATE — Perform Date/Time Functions 4-4
DKNDEQ — Dequeue on Resource Name 4-9
DKNDYNA — Dynamically Allocate/Unallocate Permanent Data Sets 4-11
DKNENQ — Enqueue on Resource Name 4-15
DKNPDSIO — Perform PDS I/O Processing 4-17
DKNQGET — Perform QSAM Input Processing 4-23
DKNQPUT — Perform QSAM Output Processing 4-28
DKNTDYNA — Dynamically Allocate/Unallocate Temporary Data Sets 4-33
DKNVSMIO — Perform VSAM I/O Processing 4-37

Glossary X-1

Bibliography X-13
ACF/VTAM Publications X-13
Document Processor Support Publications X-13
High Performance Transaction System Publications (Version 1) X-13
High Performance Transaction System Publications (Version 2) X-13
MVS Publications (Version 5) X-13
RACF Publications X-14
CPCS Enhanced System Manager Publication X-14

Index X-15

Tables

1-1. Application Task Attachment Communication Control Blocks 1-4
1-2. Open String Status Modification at Task Abend 1-7
1-3. DKNTERM2 Communication Control Blocks 1-11
1-4. DKNTERM2 Request Codes 1-12
1-5. DKNTERM2 Return Codes 1-12
1-6. DKNGETXI Communication Control Blocks 1-21
1-7. DKNGETB2 Communication Control Blocks 1-27
1-8. DKNSMSG Communication Control Blocks 1-39
1-9. DKNSMSG Return Codes 1-40
1-10. DKNMASS Communication Control Blocks 1-48
1-11. DKNMASS Operation Modes 1-49
1-12. DKNMASS Return Codes 1-49
1-13. DKNUAM Communication Control Blocks 1-57
1-14. DKNUAM Return Codes 1-58
1-15. DKNSTGD Return Codes 1-60
1-16. DKNPCTLI Communication Control Blocks 1-64
1-17. DKNPCTLI FUNCTIONS 1-64
1-18. DKNPCTLI Return Codes 1-65
1-19. DKNTGUT Communication Control Blocks 1-70
1-20. DKNTGUT FUNCTIONS 1-70
1-21. DKNTGUT Return Codes 1-70
1-22. DKNCYCI Communication Control Blocks 1-77
1-23. DKNCYCI Return Codes 1-78
1-24. DKNBCFIO Communication Control Blocks 1-81
1-25. DKNIGEN Communication Control Blocks 1-86
1-26. DKNIGEN Return Codes 1-86
1-27. DKNADCB2 Communication Control Blocks 1-90
1-28. DKNADCB2 Return Codes 1-90
1-29. DKNADCB3 Communication Control Blocks 1-97
1-30. DKNADCB3 Return Codes 1-97
1-31. Full-Page Buffering Mask Record Format 1-104
1-32. Full-Page Buffering Header/Trailer Carriage Control Codes 1-105
1-33. DKNMDXR Communication Control Blocks 1-113
1-34. DKNMDXR Return Codes 1-114
1-35. DKNMAYI Communication Control Blocks 1-120
1-36. DKNMAYI Return Codes 1-120
1-37. DKNCDIF Communication Control Blocks 1-123
1-38. DKNCDIF Return Codes 1-124
1-39. DKNMOLRI Communication Control Blocks 1-129
1-40. DKNMOLRI Routine Return Codes 1-130
1-41. DKNUEM Communication Control Blocks 1-133
1-42. DKNUEM Return/Reason Codes 1-134
2-1. ALLO Exit Communication Control Blocks 2-4
2-2. CDIF Exit Communication Control Blocks 2-7
2-3. CDIF Exit Routine Return Codes 2-8
2-4. CYCL Exit Communication Control Blocks 2-12
2-5. CYCL Exit Routine Return Codes 2-12
2-6. DFTI Exit 1 Communication Control Blocks 2-15
2-7. DFTI Exit 1 Routine Return Codes 2-16
2-8. DFTI Exit 2 Communication Control Blocks 2-19

2-9. DFTI Exit 2 Routine Return Codes 2-20
2-10. DFTO Exit Communication Control Blocks 2-24
2-11. DKNDFTP Exit Routine Parameter List 2-29
2-12. DFTP Exit Routine Return Codes 2-30
2-13. FSCN Exit Communication Control Blocks 2-34
2-14. FSCN Exit Routine Return Codes 2-34
2-15. HCDM Exit 1 Communication Control Block 2-37
2-16. HCDM Exit 2 Communication Control Blocks 2-39
2-17. HCDM Exit 2 Routine Return Codes 2-40
2-18. HCDM Exit 3 Communication Control Blocks 2-44
2-19. HCDM Exit 4 Communication Control Block 2-46
2-20. HCDM Exit 4 Routine Return Codes 2-46
2-21. IDCR Exit Communication Control Blocks 2-48
2-22. IDCR Exit Routine Return Codes 2-48
2-23. MDIS Exit Communication Control Blocks 2-51
| 2-24. MICR Exit 1 Communication Control Block 2-55
| 2-25. MICR Exit 2 Communication Control Block 2-60
| 2-26. MICR Exit 3 Communication Control Block 2-71
| 2-27. MICR Exit 4 Communication Control Block 2-77
| 2-28. MICR Exit 5/6 Communication Control Block 2-83
2-29. MOLRI Exit Communication Control Blocks 2-89
2-30. MOLRI Exit Routine Return Codes 2-89
2-31. MRGF Exit Communication Control Blocks 2-93
2-32. MTASK Exit 1 Communication Control Blocks 2-95
2-33. MTASK Exit 1 Communication Control Blocks 2-97
2-34. RCVY Exit Communication Control Blocks 2-100
2-35. SECR Exit Communication Control Blocks 2-102
2-36. SECR Exit Routine Attachment Return Codes 2-103
2-37. SECR Exit Routine SIGNON Transaction Call Return Codes . . . 2-103
2-38. VSMGR Exit 1 Communication Control Blocks 2-107
2-39. VSMGR Exit 1 Routine Return Codes 2-107
2-40. VSMGR Exit 2 Communication Control Blocks 2-110
2-41. VSMGR Exit 2 Routine Return Codes 2-110
2-42. VTASK Exit 1 Routine Entry Conventions 2-112
2-43. VTASK Exit 2 Routine Entry Conventions 2-113
2-44. VTASK Exit 3/4 Routine Entry Conventions 2-114
2-45. VTASK Exit 5 Routine Entry Conventions 2-115
4-1. DKNDATE Communication Control BLock 4-4
4-2. DKNDEQ Communication Control Blocks 4-9
4-3. DKNDYNA Communication Control Blocks 4-11
4-4. DKNDYNA Return Codes 4-11
4-5. DKNENQ Communication Control Blocks 4-15
4-6. DKNPDSIO Communication Control Blocks 4-17
4-7. DKNPDSIO Return Codes 4-17
4-8. DKNQGET Communication Control Blocks 4-23
4-9. DKNQGET Return Codes 4-23
4-10. DKNQPUT Communication Control Blocks 4-28
4-11. DKNQPUT Return Codes 4-28
4-12. DKNTDYNA Communication Control Blocks 4-33
4-13. DKNVSMIO Communication Control Blocks 4-37
4-14. DKNVSMIO non-VSAM Return Codes 4-38

Figures

1-1. Application Task Linkage Example 1 1-4
1-2. Application Task Linkage Example 2 1-6
1-3. Returning to CPCS-I COBOL Example 1 1-8
1-4. Returning to CPCS-I Assembler Example 1 1-9
1-5. DKNTERM2 Example 1 1-13
1-6. DKNTERM2 Example 2 1-17
1-7. DKNGETXI Example 1 1-22
1-8. DKNGETXI Example 2 1-24
1-9. DKNGETB2 Example 1 1-27
1-10. DKNGETB2 Example 2 1-30
1-11. Setting Up Addressability for User Executive Task 1-33
1-12. User Executive Task Searching Parameter List Extension 1-34
1-13. User Executive Task Constructing Communication Buffer TRT Table 1-35
1-14. User Executive Task Wait For Inbound Communication Buffers . . 1-36
1-15. User Executive Task Scan For Inbound Communication Buffers . . 1-36
1-16. User Executive Task Freeing Communication Buffers 1-37
1-17. DKNSMSG Example 1 1-41
1-18. DKNSMSG Example 2 1-43
1-19. DKNWTO COBOL Example 1-46
1-20. DKNMASS Example 1 1-52
1-21. DKNMASS Example 2 1-55
1-22. DKNUAM Example 1-59
1-23. DKNSTGD COBOL Example 1 1-61
1-24. DKNSTGD Assembler Example 1 1-62
1-25. DKNPCTLI Example 1 1-66
1-26. DKNPCTLI Example 2 1-68
1-27. TGUT Assembler Example 1-72
1-28. Calling TGUT COBOL Example 1-75
1-29. TGUT Processing COBOL Example 1-75
1-30. Ending TGUT COBOL Example 1-76
1-31. DKNCYCI Example 1 1-79
1-32. DKNCYCI Example 2 1-80
1-33. DKNBCFIO COBOL Example 1 1-81
1-34. DKNBCFIO Assembler Example 1 1-82
1-35. DKNLINK2 COBOL Example 1-84
1-36. DKNLINK2 Assembler Example 1-85
1-37. DKNIGEN Example 1 1-87
1-38. DKNIGEN Example 2 1-88
1-39. DKNADCB2 Example 1 1-91
1-40. DKNADCB2 Example 2 1-93
1-41. DKNADCB3 Example 1 1-98
1-42. DKNADCB3 Example 2 1-100
1-43. Columnar Report Example 1 1-106
1-44. Columnar Report Example 2 1-109
1-45. DKNMDXR Example 1 1-115
1-46. DKNMDXR Example 2 1-118
1-47. DKNMAYI Example 1 1-120
1-48. DKNMAYI Example 2 1-122
1-49. DKNCDIF Example 1 1-124
1-50. DKNCDIF Example 2 1-126

1-51. DKNMOLRI Example 1 1-131
1-52. DKNUEM Example 1-135
2-1. CDIF Exit Routine Example 1 2-5
2-2. CDIF Exit Routine Example 1 2-8
2-3. CDIF Exit Routine Example 2 2-10
2-4. CYCL Exit Routine Example 1 2-13
2-5. Assembler DFTI Exit 1 Routine Example 1 2-16
2-6. COBOL DFTI Exit 1 Routine Example 1 2-17
2-7. Assembler DFTI Exit 2 Routine Example 1 2-20
2-8. COBOL DFTI Exit 2 Routine Example 1 2-22
2-9. DFTO Exit Routine Example 1 2-24
2-10. DFTO Exit Routine Example 2 2-26
2-11. DFTP Exit Routine Example 1 2-31
2-12. FSCN Exit Routine Example 1 2-34
2-13. HCDM Exit 1 Routine Example 1 2-38
2-14. HCDM Exit 2 Routine Example 1 2-41
2-15. HCDM Exit 3 Routine Example 1 2-45
2-16. HCDM Exit 4 Routine Example 1 2-47
2-17. MDIS Exit Routine Example 1 2-52
2-18. MICR BEGIN Exit Routine Example 1 2-56
2-19. MICR BEGIN Exit Routine Example 2 2-57
2-20. MICR Document Processing Exit Routine Example 1 2-63
2-21. MICR Document Processing Exit Routine Example 2 2-66
2-22. MICR CDM Exit Example Buffer 2-69
2-23. MICR CDM Exit Routine Example 1 2-72
2-24. MICR CDM Exit Routine Example 2 2-74
2-25. MICR Exit 4 Routine Example 1 2-78
2-26. MICR Exit 5 Routine Example 1 2-84
2-27. MOLRI Exit Routine Example 1 2-90
2-28. MRGE Exit Routine Example 1 2-94
2-29. MTASK Exit 1 Routine Example 1 2-96
2-30. MTASK Exit 2 Routine Example 1 2-98
2-31. RCVY Exit Routine Example 1 2-101
2-32. SECR Exit Routine Example 1 2-104
2-33. VSMGR Exit 1 Routine Example 1 2-108
2-34. VSMGR Exit 2 Routine Example 1 2-111
3-1. ALDSN Macro Definition - Example 1 3-8
3-2. ALDSN Macro Definition - Example 2 3-9
3-3. ALLOC Macro Definition - Example 1 3-15
3-4. ALLOC Macro Definition - Example 2 3-15
3-5. CPCSNAME Macro Definition - Example 1 3-16
3-6. DKNAMODE Macro Definition - Example 1 3-20
3-7. DKNAMODE Macro Definition - Example 2 3-20
3-8. DKNAMODE Macro Definition - Example 3 3-20
3-9. DKNFILL Macro Definition - Example 1 3-22
3-10. DKNFILL Macro Expansion - Example 1 3-22
3-11. DKNREGS Macro Definition - Example 1 3-23
3-12. DKNREGS Macro Expansion - Example 1 3-23
3-13. DKNSBT Macro Definitions - Example 1 3-25
3-14. DKNXMODE Macro Definition - Example 1 3-27
3-15. DKNXMODE Macro Definition - Example 2 3-27
3-16. FFLDL Macro Definition - Example 1 3-28
3-17. FFLDL Macro Expansion - Example 1 3-29
3-18. FFLDL Macro Definition - Example 2 3-29

3-19. FFLDL Macro Expansion - Example 2 3-29
3-20. FPACK Macro Definition - Example 1 3-32
3-21. FPACK Macro Expansion - Example 1 3-32
3-22. FPACK Macro Result - Example 1 3-32
3-23. FPACK Macro Definition - Example 2 3-33
3-24. FPACK Macro Expansion - Example 2 3-33
3-25. FPACK Macro Result - Example 2 3-33
3-26. FUNPK Macro Definition - Example 1 3-35
3-27. FUNPK Macro Expansion - Example 1 3-35
3-28. FUNPK Macro Result - Example 1 3-35
3-29. FUNPK Macro Definition - Example 2 3-35
3-30. FUNPK Macro Expansion - Example 2 3-36
3-31. FUNPK Macro Result - Example 2 3-36
3-32. XRETURN Macro Definition - Example 1 3-37
3-33. XRETURN Macro Definition - Example 2 3-38
3-34. XRETURN Macro Definition - Example 3 3-38
4-1. DKNDATE Subroutine Example 1 4-4
4-2. DKNDATE Subroutine Example 2 4-6
4-3. DKNDEQ Subroutine Example 1 4-9
4-4. DKNDYNA Example 1 4-12
4-5. DKNDYNA Example 2 4-13
4-6. DKNENQ Subroutine Example 1 4-15
4-7. DKNPDSIO Subroutine Example 1 4-18
4-8. DKNPDSIO Subroutine Example 2 4-20
4-9. DKNQGET Subroutine Example 1 4-24
4-10. DKNQGET Subroutine Example 2 4-26
4-11. DKNQPUT Subroutine Example 1 4-29
4-12. DKNQPUT Subroutine Example 2 4-30
4-13. DKNTDYNA COBOL Example 1 4-34
4-14. DKNTDYNA Assembler Example 1 4-35
4-15. DKNVSMIO Subroutine Example 1 4-40
4-16. DKNVSMIO Subroutine Example 2 4-43

Notices

This reference is intended to help customers perform modifications of the IBM Check Processing Control System International MVS/ESA (CPCS-I) licensed program. This reference primarily documents Product-sensitive Programming Interface and Associated Guidance Information provided by CPCS-I.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of CPCS-I. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

However, this reference also documents General-use Programming Interface and Associated Guidance Information and Diagnosis, Modification, or Tuning Information provided by CPCS-I.

General-Use programming interfaces allow the customer to write programs that obtain the services of CPCS-I.

Diagnosis, Modification, or Tuning Information is provided to help the customer to do customization, modification, or tuning of CPCS-I.

Attention: Do not use this Diagnosis, Modification, or Tuning Information as a programming interface.

IBM recommends that customers use the external interfaces when modifications of CPCS-I are desired. IBM can implement additional CPCS-I external interfaces at customer request, usually for a fee. With the exception of “user reserved” labeled areas, CPCS-I source code modifications will be made at the user's own risk.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM’s intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user’s responsibility.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact: IBM Corporation, Department MG39/201, 8501 IBM Drive, Charlotte, NC 28262-8563, U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York, 10504-1785, U.S.A.

Year 2000 Readiness

This IBM program, when used in accordance with its associated documentation, is capable of correctly processing, providing and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with this IBM program properly exchange accurate date data with it.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM®, ACF/VTAM®, AD/Cycle®, COBOL/370™, ImagePlus®, MVS/ESA™, RACF™, SAA®, VTAM®.

Other company, product, and service names may be trademarks or service marks of others.

About This Publication

This reference supplies programming information for personnel who perform modifications of the IBM Check Processing Control System (CPCS-I) licensed program.

This reference is intended to be used with the *CPCS-I Programming Guide*.

Who Should Read This Publication?

This book is for system programmers, application programmers and operational personnel that install and maintain CPCS-I systems.

How Is This Publication Organized?

This book contains the following chapters:

Chapter 1, “CPCS-I Application Programming” contains information about application programming under CPCS-I.

Chapter 2, “CPCS-I User Exit Programming” documents the CPCS-I user exits.

Chapter 3, “Assembler Macros Provided by CPCS-I” documents CPCS-I macros that can be included in CPCS-I application programs and exit routines.

Chapter 4, “Subroutines Provided by CPCS-I” documents CPCS-I subroutines that can be included in CPCS-I application programs and exit routines.

This book also contains a glossary, a bibliography, and an index.

Related Publications

The following publications contain information that relates to Check Processing Control System International MVS/ESA (CPCS-I). For an additional list of relevant publications, see the “Bibliography.”

- *IBM Check Processing Control System International MVS/ESA: General Information*, GC31-2944
  Short Title: *CPCS-I General Information*

  This publication gives a general introduction to CPCS-I. It describes various features and advantages of CPCS-I and the hardware and software requirements for operating this system. It also discusses CPCS-I support of the IBM 3890 Document Processor and the IBM 3890/XP Series document processors, along with some of the features of these processors.

- *IBM Check Processing Control System International MVS/ESA: Installation Guide*, GC31-2942
  Short Title: *CPCS-I Installation Guide*

  This guide describes the steps necessary for using the IBM System Modification Program Extended (SMP/E) procedures to install CPCS-I software. It also provides installation procedures for generating CPCS-I modules and

creating operational data sets. It provides data for sample problems to test and verify operations after CPCS-I installation.

- *IBM Check Processing Control System International MVS/ESA: Terminal Operations Guide*, SC31-2946
  Short Title: *CPCS-I Terminal Operations Guide*

  This guide explains how to perform CPCS-I tasks and is written for the CPCS-I operators. Included in this guide are terminal operations for the MICR restart procedures and sample reports.

- *IBM Check Processing Control System International MVS/ESA: Programming Guide*, SC31-2948
  Short Title: *CPCS-I Programming Guide*

  This guide contains guidelines for CPCS-I programmers. It includes information about application-program processing, problem analysis and documentation procedures.

- *IBM Check Processing Control System International MVS/ESA: Programming Reference*, GC31-3997
  Short Title: *CPCS-I Programming Reference*

  This publication gives a structured view of the CPCS-I interfaces, specifically application programming, Assembler macros, subroutines, and some control block information.

- *IBM Check Processing Control System International MVS/ESA: Customization Guide*, SC31-2943
  Short Title: *CPCS-I Customization Guide*

  This guide provides customization information for CPCS-I programmers. It also includes system programming information, and generation and installation procedures.

- *IBM Check Processing Control System International MVS/ESA: Messages and Codes*, SC31-3981
  Short Title: *CPCS-I Messages and Codes*

  This book describes console and supervisor messages, as well as program return and exit codes.

- *IBM Check Processing Control System International MVS/ESA: Propagation of Adjustments*, SC31-3994
  Short Title: *CPCS-I Propagation of Adjustments Guide*

  This guide contains the guidelines for the CPCS-I personnel who use the Propagation of Adjustments (PRAD) feature. It includes functional descriptions and information about terminal operations, programming, and application output.

- *IBM Check Processing Control System International MVS/ESA: Master Index*, SC31-3980
  Short Title: *CPCS-I Master Index*

  This reference combines the index entries for all the publications in the CPCS-I library.

Summary of Changes for SC31-3997-03.

Enhanced Prime: CPCS-I supports the capture of multiple entries on a single prime pass without the use of subsets.

ALLO User Exit: The ALLO user exit allows additional document processor allocation/unallocation request and user validations.

Chapter 1. CPCS-I Application Programming

Overview 1-3
Adding CPCS-I User Tasks 1-3
Accessing the Start Parameters 1-4
Returning to CPCS-I 1-7
Performing Terminal I/O 1-11
Determining Route Codes 1-21
Sending Messages to User Executive Tasks 1-27
Writing User Executive Tasks 1-33
Issuing Messages 1-39
Performing String I/O 1-47
Processing User Areas with the User Area Manager 1-57
Deleting Strings 1-60
Accessing/Updating Pass-to-Pass Control Information 1-64
Using the Tracer Group Update Utility 1-70
Accessing/Updating Cycle Control Information 1-77
Accessing Bank Control Information 1-81
Finding the Next Available Kill-Bundle Record 1-84
Obtaining Unique Sequence Numbers 1-86
Generating Spooled Reports 1-90
Generating JES Reports 1-97
Generating Columnar Reports 1-104
Generating Expanded Codeline Record Reports 1-113
Performing Security Checks 1-120
Creating DFTI Input Data Sets 1-123
Interfacing with Host Codeline Edit Routines 1-129
Invoking User Exits With the User Exit Manager 1-133

Overview

This chapter supplies information necessary for writing CPCS-I application programs and user exits.

Use this information in conjunction with the detailed information supplied in the CPCS-I copybooks.

Adding CPCS-I User Tasks

There are three types of user tasks which may be added to CPCS-I:

- Application Tasks
- CIMS/RIC Application Tasks
- User Executive Tasks

Application tasks may be written to perform check processing functions. They may use a wide variety of CPCS-I services and resources. Application tasks may be started manually from a CPCS-I terminal, or automatically by the Enhanced System Manager.

CIMS/RIC application tasks have the added capacity to perform calls to the CIMS/RIC interface and process information from the CIMS/RIC database. Both CIMS and RIC application tasks are defined to CPCS-I by adding the CIMS=1 parameter to the task's APCB entry. See the appropriate CIMS or RIC manuals for information about coding the application tasks themselves.

User executive tasks have the added capacity of being able to remain loaded and execute for the entire duration of CPCS-I. They are defined to CPCS-I by adding the EXTSK=Y parameter to the task's APCB entry. A more detailed description of user executive tasks may be found in the *CPCS-I Programming Guide* and the *CPCS-I Customization Guide*.

To add a user task to CPCS-I, you must create the application program and inform CPCS-I of the load module name, its CPCS-I characteristics, and its resource needs.

A user task is defined to CPCS-I by adding the task's APCB entry to the DKNBLDL member, in the CPCSI.V01R01.SDKNSRC1 PDS (see "Describing an Application Task's Environment" in the *CPCS-I Customization Guide*, for information on the APCB macro and the DKNBLDL table).

Note: The APCBs must be in alphabetic sequence by load module name in the DKNBLDL table.

Once updated, the DKNBLDL member must be reassembled and CPCS-I must be restarted to permanently define/re-define a task. The task definitions may be temporarily altered (until the next CPCS-I startup) while CPCS-I is running, with the DKNBLDL online editor task (BLDE). See the *CPCS-I Terminal Operations Guide* for a description of how to use the BLDE task.

Accessing the Start Parameters

When CPCS-I attaches a manually-started application task, it passes to it the following two parameters:

Table 1-1. Application Task Attachment Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB

Application Task Start Parameters

- Start parameters - 56 bytes of whatever additional information the terminal operator had entered when he starting the task.
- Generation parameters - 8 bytes of miscellaneous system and terminal information.
- Operator ID - 4 bytes of operator ID. An operator ID of binary zeroes means the task was auto-started by Enhanced System Manager.

The standard MVS/ESA* linkage conventions are followed.

Restrictions

Start parameters are present only for manually-started tasks. If a task was auto-started by Enhanced System Manager, then the Operator ID will be binary zeroes, and the Enhanced System Manager control blocks must be inspected for the task's start parameters. Please see the *CPCS Enhanced System Manager User's Guide* for more information.

Example 1

Perform two different functions based on a three-character task start parameter.

COBOL Code:

```
     ...................
     ...................
     ...................
/
 LINKAGE SECTION.

 COPY DKNCATCB.
```

Figure 1-1. Application Task Linkage Example 1

* Trademark of IBM

```
 01  APPLICATION-PARMS.
     05  AP-START-PARMS.
         10  APS-PARM             PIC   X(03).
             88  APS-FUNCTION-1             VALUE '001'.
             88  APS-FUNCTION-2             VALUE '002'.
         10  FILLER               PIC   X(53).
     05  AP-GENERATION-PARMS      PIC   X(08).
     05  AP-OPERATOR-ID           PIC   X(04).
/
 PROCEDURE DIVISION               USING APTCB
                                        APPLICATION-PARMS.

     EVALUATE TRUE

         WHEN APS-FUNCTION-1
              PERFORM 1000-PROCESS-FUNCTION-1

         WHEN APS-FUNCTION-2
              PERFORM 2000-PROCESS-FUNCTION-2

         WHEN OTHER
              PERFORM 3000-PROCESS-INVALID-PARM

     END-EVALUATE

 1000-PROCESS-FUNCTION-1.

     ...................
     ...................
     ...................

 2000-PROCESS-FUNCTION-2.

     ...................
     ...................
     ...................

 3000-PROCESS-INVALID-PARM.

     ...................
     ...................
     ...................
```

Figure 1-1. Application Task Linkage Example 1

Example 2

Accept a start parameter consisting of a one-character cycle ID and a four-character entry number separated by any character.

Assembler Code:

```
         .................
         .................
         .................
*
         L     R4,0(R1)               GET APTCB @
         USING APTCB,R4               BASE APTCB
         L     R5,4(R1)               GET APPLICATION PARMS @
         USING PARMS,R4               BASE APPLICATION PARMS
*
         COPY  APTCB                  APPLICATION TASK CONTROL BLOCK
*
PARMS    DSECT                        APPLICATION PARMS
STRTPRMS DS   0CL56                   . START PARAMETERS
STCYCLE  DS    CL1                      - CYCLE ID
         DS    CL1                      - SEPARATOR
STENTRY  DS    CL4                      - ENTRY NUMBER
         DS    CL50                     - FILLER
GENRPRMS DS    CL8                    . GENERATION PARAMETERS
OPERTRID DS    CL4                    . OPERATOR ID
         .................
         .................
         .................
```

Figure 1-2. Application Task Linkage Example 2

Returning to CPCS-I

To prevent main storage from fragmenting, assembler applications must issue the FREEPOOL macro after closing any QSAM data set, before they return to CPCS-I. Assembler applications should also issue a FREEMAIN/STORAGE RELEASE macro for the private storage dynamically allocated by the application. Main storage fragmentation could eventually degrade or cripple CPCS-I processing.

Applications should set register 15 (assembler programs) or the RETURN-CODE special register (COBOL programs) to zero or, for some conditions to a meaningful code, before returning to CPCS-I. The return code must have a maximum length of three hexadecimal digits.

If the return code is not zero, message `AAAAAAAA ENDED UCCC TTT OOO`, where AAAAAAAA is the task name, CCC is the application's return code in hexadecimal, TTT is the terminal ID (manually started tasks), and OOO is the operator ID (manually started tasks), is displayed on the CPCS-I supervisor terminal and sent to the scroll log. The task's DETACH message, in the CPCS-I log, also includes the return code in the form `DETACH AAAAAAAA S000 UCCC`.

Notes:

1. Tasks defined as depending on previous task executions in Enhanced System Manager work flows, are not automatically started until all previous tasks in the work flow end with a return code of zero.
2. DKNSMSG (see "Issuing Messages" on page 1-39) may be called to issue a supervisor message when a task terminates with a non-zero completion code.

User-abended tasks are treated as non-zero return-code-ended tasks ending with a return code equal to the user abend code.

System-abended tasks are also treated as non-zero return-code ended tasks, but ENDED gets replaced by ABEND, and UCCC by SCCC, where CCC is the system abend code in this case, in the task abnormal completion message. The CPCS-I log task detach message has in this case the form `DETACH AAAAAAAA SCCC U000`.

When a task abends, CPCS-I prints the related spool data sets and, if attached, releases the terminal. Also, CPCS-I modifies the status of open strings as follows:

Table 1-2 (Page 1 of 2). Open String Status Modification at Task Abend

String Type	String Status	
	Before Abend	After Abend
I	Input	Closed
	Output	Restart
D	Input	Closed
	Output	Purged
R	Input	Closed
	Output	Restart
M	Input	Closed
	Output	Purged

Table 1-2 (Page 2 of 2). Open String Status Modification at Task Abend

String Type	String Status	
	Before Abend	After Abend
W	Input	Closed
	Output	Purged

Example 1

Stop processing and transmit the LOAD/DKNMASS return code to CPCS-I when loading or calling DKNMASS is not successful (see “Performing String I/O” on page 1-47 for information on DKNMASS).

COBOL Code:

```
       ID DIVISION.
       PROGRAM-ID. TSRET.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
      /
       WORKING-STORAGE SECTION.

       01  COMPLETION-CODE.
           05  FILLER                    PIC  X(01) VALUE X'00'.
           05  CC-HEX                    PIC  X(01).
       01  FILLER                        REDEFINES  COMPLETION-CODE.
           05  CC-BIN                    PIC  9(01) COMP SYNC.

       COPY DKNCRZA.

       COPY DKNCRZE2.

       COPY DKNCRDI                       REPLACING ==:DI:==
                                          BY        ==DI==.

       COPY DKNCRZD                       REPLACING ==:ZD:==
                                          BY        ==ZD==.
           ...................
           ...................
           ...................
      /
       LINKAGE SECTION.

       COPY DKNCATCB.

       01   START-PARMS                   PIC  X(68).
      /
       PROCEDURE DIVISION                 USING APTCB
```

Figure 1-3 (Part 1 of 2). Returning to CPCS-I COBOL Example 1

```
                                          START-PARMS.

          ...................
          ...................
          ...................

          CALL 'DKNMASS'                USING ZA
                                              ZE
                                              DI
                                              ZD.

          IF  NOT ZA-NO-ERROR
              MOVE ZA-RETURN-CODE       TO CC-HEX
              MOVE CC-BIN               TO RETURN-CODE
              GOBACK.

          ...................
          ...................
          ...................
```

Figure 1-3 (Part 2 of 2). Returning to CPCS-I COBOL Example 1

Assembler Code:

```
TSRET    CSECT
TSRET    AMODE 31
TSRET    RMODE ANY
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         ................
         ................
         ................
*
         LOAD  EPLOC=DKNMASS
         LTR   R15,R15               SUCCESSFUL LOAD?
         BNZ   RETURN                N. EXIT PROGRAM
         LR    R15,R0                GET DKNMASS LOAD @
         LA    R1,PARMS              GET DKNMASS PARM LIST @
         BASSM R14,R15               CALL DKNMASS
         CLI   ZARETCD,RCGOOD        SUCCESSFUL OPERATION ?
         BNE   MASSERR               N. PROCESS DKNMASS ERROR
*
         ................
         ................
         ................
*
```

Figure 1-4 (Part 1 of 2). Returning to CPCS-I Assembler Example 1

```
MASSERR  DS   0H                       PROCESS DKNMASS ERROR
         XR    R15,R15
         ICM   R15,B'0001',ZARETCD     TRANSMIT DKNMASS RETURN CODE
         B     RETURN                  EXIT PROGRAM
*
         ..............
         ..............
         ..............
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
RETCDES  DS  0H                        DKNMASS RETURN CODES
RCGOOD   EQU  C'0'                     . SUCCESSFUL COMPLETION
*
PARMS    DS  0F                        DKNMASS PARM LIST
         DC   A(ZA)                    . CALL PARMS @
         DC   A(ZE)                    .
         DC   A(DI)                    . COMPRESSED CODELINE RECD @
         DC   A(ZD)                    . UNCOMPRESSED CODELINE RECD @
*
         COPY ZADSCT                   DKNMASS CALL PARMS
*
         COPY ZEDSCT                   DKNMASS DATA PARMS
*
         COPY DIDSCT                   COMPRESSED CODELINE RECORD
*
         COPY ZDDSCT                   UNCOMPRESSED CODELINE RECORD
*
         END
```

Figure 1-4 (Part 2 of 2). Returning to CPCS-I Assembler Example 1

Performing Terminal I/O

An application task that was manually started from a terminal can write messages and screens to that terminal. It can also obtain further input. Both these functions are performed by the CPCS-I graphics interface module, DKNTERM2.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNTERM2:

Table 1-3. DKNTERM2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
DKNTERM2 Call Parameters • This area contains the fields: – Request Code: 2 bytes (see valid request codes below). – Write Length: 2 binary bytes. – Read Length: 2 binary bytes. The application task should set this field to indicate the size of its input area prior to calling DKNTERM2. DKNTERM2 resets this field to indicate the actual number of characters read. – Return Code: 1 character (see return codes below).		
Output Area • This area contains 3270 control characters, orders, addresses/attributes, and the text to be displayed. It is used on 'write' requests.		
Input Area • This area contains the 3270 attention identifier (1 byte), the cursor address (2 bytes), the 'set buffer address' order (3 bytes), and the operator's response (variable length). It is used on 'read' requests.		

Note: For information on how to build 3270 data streams, see:

- *3270 Information Display System - 3274 Control Unit Reference Summary*
- *3274 Control Unit Description and Programming Guide*
- *3270 Information Display System Data Stream Programmer's Reference*.

Valid request codes are:

Table 1-4. DKNTERM2 Request Codes

Code (hex)	Description
000A	Erase, write to line address, read.
000B	Read.
000C	Write
000D	Erase, write.
000E	Write, read.
000F	Erase, write, read.
0010	Release terminal.
0011	Write to line address.
0012	Erase, write to line address.
0013	Write to line address, read.

Note: The 'release terminal' function lets an application task release its attached terminal when it no longer needs terminal functions, allowing the operator to start another task while the first one continues processing. A task that does not have a terminal available to it has binary zeros/low values in the APTCB terminal pointer (assembler TCBTERM, and COBOL APTCB-TERM) fields.

DKNTERM2 may return the following completion codes in the call parameters control block:

Table 1-5. DKNTERM2 Return Codes

Code (dec)	Description
+00	Successful completion.
+01	Permanent terminal I/O error.
+03	Buffer length error.
+04	Invalid request. • This completion code could be the result of an invalid operation code, a zero read length (read requests), a zero write length (write requests), or an invalid line address.

Restrictions

DKNTERM2 should not be called by a task that does not have a terminal available to it. Any attempt to do so causes the task to abend.

Example 1

Operation:

Create a CPCS-I task that is to be manually started from 24 x 80 screen terminals.

The task:

1. Displays the following screen:

```
                        APPLICATION A


ENTER CYCLE ID ==>
OR PRESS PF3 TO EXIT
```

2. Accepts a one-character cycle ID and the program function key 3. The message `<--- INVALID CYCLE` is displayed high-lighted on the response line when an invalid cycle number is entered. Lower-case alphabetic characters are converted to upper-case characters. The cycle, which must be active, is validated, and its status is obtained, calling module DKNCYCI.
3. Displays the message `TERMINAL RELEASED`, releases the terminal, and then performs the task's function, if a valid cycle number is entered.

Note: See "Accessing/Updating Cycle Control Information" on page 1-77 for information on module DKNCYCI.

COBOL Code:

```
ID DIVISION.
PROGRAM-ID. TSTERM2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  CALL-PARMS.
    05  CP-REQUEST-CODE        PIC  X(02).
    05  CP-WRITE-LENGTH        PIC S9(04)             COMP.
    05  CP-READ-LENGTH         PIC S9(04)             COMP.
    05  CP-ERROR-CODE          PIC  X(01) VALUE '0'.
        88  CP-SUCCESSFUL-TERMINAL-I-O    VALUE '0'.
        88  CP-TERMINAL-I-O-ERROR         VALUE '1'.
        88  CP-BUFFER-LENGTH-ERROR        VALUE '3'.
        88  CP-INVALID-REQUEST            VALUE '4'.

01  BUFFER-LENGTHS.
    05  BL-WRITE-LENGTH-1      PIC S9(04) VALUE +124  COMP.
    05  BL-WRITE-LENGTH-2      PIC S9(04) VALUE +023  COMP.
    05  BL-READ-LENGTH-1       PIC S9(04) VALUE +007  COMP.
    05  BL-READ-LENGTH-2       PIC S9(04) VALUE +006  COMP.
```

Figure 1-5 (Part 1 of 4). DKNTERM2 Example 1

```
 01  REQUEST-CODES.
     05  RC-ERASE-WRITE         PIC  X(02) VALUE X'000D'.
     05  RC-WRITE-READ          PIC  X(02) VALUE X'000E'.
     05  RC-ERASE-WRITE-READ    PIC  X(02) VALUE X'000F'.
     05  RC-RELEASE-TERMINAL    PIC  X(02) VALUE X'0010'.

 01  WRITE-AREA-1.
     05  WA1-RESTORE-KEYBRD-MDT PIC  X(01) VALUE X'C3'.
     05  WA1-ROW01-COLUMN01     PIC  X(03) VALUE X'114040'.
     05  WA1-PROTECTED          PIC  X(02) VALUE X'1D60'.
     05  WA1-ROW01-TEXT1        PIC  X(31) VALUE ' '.
     05  WA1-ROW01-TEXT2        PIC  X(12) VALUE 'APPLICATION'.
     05  WA1-ROW01-TEXT3        PIC  X(01) VALUE 'A'.
     05  WA1-ROW05-COLUMN01     PIC  X(04) VALUE X'11C540'.
     05  WA1-ROW05-TEXT1        PIC  X(06) VALUE 'ENTER'.
     05  WA1-ROW05-TEXT2        PIC  X(09) VALUE 'CYCLE ID'.
     05  WA1-ROW05-TEXT3        PIC  X(04) VALUE '===>'.
     05  WA1-UNPROTECTED-HILITE PIC  X(02) VALUE X'1DC9'.
     05  WA1-CURSOR-STOP        PIC  X(01) VALUE X'13'.
     05  WA1-ROW05-COLUMN22     PIC  X(03) VALUE X'11C5D6'.
     05  WA1-PROTECTED          PIC  X(02) VALUE X'1D68'.
     05  WA1-ROW05-MESSAGE      PIC  X(18) VALUE ' '.
     05  WA1-ROW06-COLUMN01     PIC  X(03) VALUE X'11C650'.
     05  WA1-PROTECTED          PIC  X(02) VALUE X'1D60'.
     05  WA1-ROW06-TEXT1        PIC  X(13) VALUE 'OR PRESS PF3'.
     05  WA1-ROW06-TEXT2        PIC  X(07) VALUE 'TO EXIT'.

 01  WRITE-AREA-2.
     05  WA2-RESTORE-KEYBRD-MDT PIC  X(01) VALUE X'C3'.
     05  WA2-R01C01-PROT        PIC  X(05) VALUE X'1140401D60'.
     05  WA2-R01-TEXT1          PIC  X(09) VALUE 'TERMINAL'.
     05  WA2-R01-TEXT2          PIC  X(08) VALUE 'RELEASED'.

 01  ERROR-MESSAGE.
     05  FILLER                 PIC  X(13) VALUE '<--- INVALID'.
     05  FILLER                 PIC  X(05) VALUE 'CYCLE'.

 01  READ-AREA.
     05  RA-ATTENTION-ID        PIC  X(01).
         88  RA-PF03                       VALUE X'F3'.
     05  RA-CURSOR-POSITION     PIC  X(02).
     05  RA-SET-BUFFFER-ADDRESS PIC  X(01).
     05  RA-SBA-ADDRESS-1       PIC  X(01).
     05  RA-SBA-ADDRESS-2       PIC  X(01).
     05  RA-RESPONSE            PIC  X(01).

 01  RESPONSE-LENGTH            PIC S9(04) VALUE +1.

 COPY DKNCCYCI.
/
 LINKAGE SECTION.
```

Figure 1-5 (Part 2 of 4). DKNTERM2 Example 1

```
 COPY DKNCATCB.

 01  TCBSPARM                  PIC  X(68).
/
 PROCEDURE DIVISION                 USING APTCB
                                          TCBSPARM.

     MOVE RC-ERASE-WRITE-READ       TO CP-REQUEST-CODE.
     SET  CYCI-STATUS-DEACTIVE      TO TRUE.
     PERFORM 1000-GET-CYCLE
         UNTIL CYCI-STATUS-ACTIVE
         OR    RA-PF03
         OR    NOT CP-SUCCESSFUL-TERMINAL-I-O.

     IF  CP-SUCCESSFUL-TERMINAL-I-O
         IF  NOT RA-PF03
             PERFORM 2000-RELEASE-TERMINAL

             IF  CP-SUCCESSFUL-TERMINAL-I-O
                 PERFORM 3000-PROCESS-APPLICATION.

     GOBACK.

 1000-GET-CYCLE.

     MOVE BL-WRITE-LENGTH-1         TO CP-WRITE-LENGTH.
     MOVE BL-READ-LENGTH-1          TO CP-READ-LENGTH.

     CALL 'DKNTERM2'                USING APTCB
                                          CALL-PARMS
                                          WRITE-AREA-1
                                          READ-AREA.

     IF  CP-SUCCESSFUL-TERMINAL-I-O
         IF  NOT RA-PF03
             PERFORM 1500-VALIDATE-CYCLE

             IF  CYCI-RETURN-GOOD
                 IF  NOT CYCI-STATUS-ACTIVE
                     MOVE ERROR-MESSAGE    TO WA1-R05-MESSAGE
                     MOVE RC-WRITE-READ    TO CP-REQUEST-CODE
                 END-IF
             END-IF
             MOVE CYCI-RETURN-CODE   TO RETURN-CODE.

 1500-VALIDATE-CYCLE.

     IF  RA-RESPONSE                ALPHABETIC-LOWER
         MOVE FUNCTION UPPER-CASE(RA-RESPONSE) TO RA-RESPONSE
```

Figure 1-5 (Part 3 of 4). DKNTERM2 Example 1

```
      SET CYCI-REQUEST-DATE          TO TRUE.
      MOVE RA-RESPONSE               TO CYCI-CYCLE-ID.

      CALL 'DKNCYCI'                 USING APTCB
                                           DKNCYCI-PARMS.

  2000-RELEASE-TERMINAL.

      MOVE RC-ERASE-WRITE            TO CP-REQUEST-CODE.
      MOVE BL-WRITE-LENGTH-2         TO CP-WRITE-LENGTH.
      MOVE BL-READ-LENGTH-2          TO CP-READ-LENGTH.

      CALL 'DKNTERM2'                USING APTCB
                                           CALL-PARMS
                                           WRITE-AREA-2
                                           READ-AREA.

      IF  CP-SUCCESSFUL-TERMINAL-I-O
          MOVE RC-RELEASE-TERMINAL   TO CP-REQUEST-CODE.

          CALL 'DKNTERM2'            USING APTCB
                                           CALL-PARMS
                                           WRITE-AREA-2
                                           READ-AREA.

      MOVE CP-ERROR-CODE             TO RETURN-CODE.

  3000-PROCESS-APPLICATION.

      ...................
      ...................
          ...................
```

Figure 1-5 (Part 4 of 4). DKNTERM2 Example 1

Example 2

Operation:

Create a CPCS-I task that is to be manually started from 24 x 80 screen terminals. The task:

1. Displays the following menu:

```
                        APPLICATION B


- OPTION 1

- OPTION 2

- OPTION 3




PF1 - HELP
PF3 - EXIT
```

2. Calls module APPLBH if program function key 1 is pressed, displays the message APPLICATION B TERMINATED and terminates if program function key 3 is pressed, calls module APPLB1 if option 1 is selected, calls module APPLB2 if option 2 is selected, and calls module APPLB3 if option 3 is selected. Selections are made tabbing to the desired option's description and pressing the enter key or any program function key. The menu is redisplayed once a selected option has completed.

Assembler Code:

```
TSTERM2  CSECT
TSTERM2  AMODE 24
TSTERM2  RMODE 24
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
*
DSPBMENU MVC   CPREQCDE,RCEWR       SET 'ERASE-WRITE-READ' REQUEST
         LA    R4,WRITBUF1          GET WRITE BUFFER @
         ST    R4,WRITBUF@          SET WRITE BUFFER @
         LA    R4,WRBF1LEN          GET WRITE BUFFER LENGTH
         STH   R4,CPWRLEN           SET WRITE BUFFER LENGTH
         LA    R4,RDBUFLEN          GET READ BUFFER LENGTH
         STH   R4,CPRDLEN           SET READ BUFFER LENGTH
         LA    R1,PARMS             GET GRAPHICS INTERFACE PARMS @
         L     R15,TERM2@           GET GRAPHICS INTERFACE @
         BASSM R14,R15              CALL GRAPHICS INTERFACE
         CLI   CPERRCDE,CPIOOK      SUCCESSFUL TERMINAL I/O ?
         BNE   TERMIOER             N. PROCESS ERROR CODE
*
         CLI   RBATTID,RBPF01       HELP REQUESTED ?
         BNE   EXITTST              N. CHECK FOR PF03
         L     R15,APPLBH@          GET APPLICATION B HELP @
         BASSM R14,R15              CALL APPLICATION B1
         B     DSPBMENU             REDISPLAY APPLICATION B MENU
*
```

Figure 1-6 (Part 1 of 4). DKNTERM2 Example 2

```
EXITTST  CLI   RBATTID,RBPF03        EXIT REQUESTED ?
         BNE   OPT1TST               N. CHECK FOR OPTION 1
         MVC   CPREQCDE,RCERWR       SET 'ERASE-WRITE' REQUEST
         LA    R4,WRITBUF2           GET WRITE BUFFER @
         ST    R4,WRITBUF@           SET WRITE BUFFER @
         LA    R4,WRBF2LEN           GET WRITE BUFFER LENGTH
         STH   R4,CPWRLEN            SET WRITE BUFFER LENGTH
         LA    R1,PARMS              GET GRAPHICS INTERFACE PARMS @
         L     R15,TERM2@            GET GRAPHICS INTERFACE @
         BASSM R14,R15               CALL GRAPHICS INTERFACE
         CLI   CPERRCDE,CPIOOK       SUCCESSFUL TERMINAL I/O ?
         BNE   TERMIOER              N. PROCESS ERROR CODE
         SR    R15,R15               CLEAR RETURN CODE REGISTER
         B     RETURN                EXIT PROGRAM
*
OPT1TST  CLI   RBCURROW,RBCURR05     OPTION 1 SELECTED ?
         BNE   OPT2TST               N. CHECK FOR OPTION 2
         L     R15,APPLB1@           GET OPTION 1 ROUTINE @
         BASSM R14,R15               CALL OPTION 1 ROUTINE
         B     DSPBMENU              REDISPLAY APPLICATION B MENU
*
OPT2TST  CLI   RBCURROW,RBCURR07     OPTION 2 SELECTED ?
         BNE   OPT3TST               N. OPTION 3 SELECTED
         L     R15,APPLB2@           GET OPTION 2 ROUTINE @
         BASSM R14,R15               CALL OPTION 2 ROUTINE
         B     DSPBMENU              REDISPLAY APPLICATION B MENU
*
OPT3TST  L     R15,APPLB3@           GET OPTION 3 ROUTINE @
         BASSM R14,R15               CALL OPTION 3 ROUTINE
         B     DSPBMENU              REDISPLAY APPLICATION B MENU
*
TERMIOER SR    R15,R15               CLEAR RETURN CODE REGISTER
         ICM   R15,B'0001',CPERRCDE  PASS GRAPHICS RETURN CODE
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
SAVEAREA DS  18F                     OUR SAVE AREA
*
PARMS    DS   0F                     DKNTERM2 PARM LIST
APTCB@   DS    A                     . APTCB @
         DC    A(CALLPRMS)           . CALL PARMS @
WRITBUF@ DS    A                     . WRITE AREA @
         DC    A(READBUFF)           . READ AREA @
*
```

Figure 1-6 (Part 2 of 4). DKNTERM2 Example 2

```
CALLPRMS DS   0H                    DKNTERM2 CALL PARMS
CPREQCDE DS    XL2                  . REQUEST CODE
CPWRLEN  DS    XL2                  . WRITE LENGTH
CPRDLEN  DS    XL2                  . READ LENGTH
CPERRCDE DS    C                    . ERROR CODE
CPIOOK   EQU   C'0'                   - SUCCESSFUL TERMINAL I/O
CPIOERR  EQU   C'1'                   - TERMINAL I/O ERROR
CPLENERR EQU   C'3'                   - BUFFER LENGTH ERROR
CPINVREQ EQU   C'4'                   - INVALID REQUEST
*
REQCODES DS   0H                    REQUEST CODES
RCEWR    DC    X'000F'              . ERASE WRITE READ
RCERWR   DC    X'000D'              . ERASE WRITE
*
WRITBUF1 DS   0H                    WRITE BUFFER 1
         DC    X'C3'                . RESTORE KEYBOARD & MDT'S
         DC    X'114040'            . ROW 01, COLUMN 01
         DC    X'1D60'                - PROTECTED
         DC  31C' '                   - TEXT
         DC    C'APPLICATION B'       - TEXT
         DC    X'11C540'            . ROW 05, COLUMN 01
         DC    X'1DC9'                - UNPROTECTED
         DC    X'13'                  - CURSOR
         DC    C'-'                   - TEXT
         DC    X'1D60'                - PROTECTED
         DC    C'OPTION 1'            - TEXT
         DC    X'11C760'            . ROW 07, COLUMN 01
         DC    X'1DC9'                - UNPROTECTED
         DC    C'-'                   - TEXT
         DC    X'1D60'                - PROTECTED
         DC    C'OPTION 2'            - TEXT
         DC    X'114A40'            . ROW 09, COLUMN 01
         DC    X'1DC9'                - UNPROTECTED
         DC    C'-'                   - TEXT
         DC    X'1D60'                - PROTECTED
         DC    C'OPTION 3'            - TEXT
         DC    X'115B60'            . ROW 23, COLUMN 01
         DC    X'1D60'                - PROTECTED
         DC    C'PF1 - HELP'          - TEXT
         DC    X'115CF0'            . ROW 24, COLUMN 01
         DC    X'1D60'                - PROTECTED
         DC    C'PF3 - EXIT'          - TEXT
WRBF1LEN EQU   *-WRITBUF1            WRITE BUFFER 1 LENGTH
*
WRITBUF2 DS   0H                    WRITE BUFFER 2
         DC    X'C3'                . RESTORE KEYBOARD & MDT'S
         DC    X'114040'            . ROW 01, COLUMN 01
         DC    X'1D60'                - PROTECTED
         DC    C'APPLICATION B '      - TEXT
         DC    C'TERMINATED'          - TEXT
WRBF2LEN EQU   *-WRITBUF2            WRITE BUFFER 2 LENGTH
*
```

Figure 1-6 (Part 3 of 4). DKNTERM2 Example 2

```
READBUFF DS   0H                    READ BUFFER
RBATTID  DS    X                    . ATTENTION KEY ID
RBPF01   EQU   X'F1'                  - PROGRAM FUNCTION KEY 1
RBPF03   EQU   X'F3'                  - PROGRAM FUNCTION KEY 3
RBCURLOC DS   0XL2                  . CURSOR LOCATION
RBCURROW DS    X                      - ROW
RBCURR05 EQU   X'C5'                     . ROW 5
RBCURR07 EQU   X'C7'                     . ROW 7
         DS    X                      - COLUMN
         DS    X                    . 'SET BUFFER ADDRESS' ORDER
         DS    X                    . SBA LOCATION 1
         DS    X                    . SBA LOCATION 2
         DS    X                    . OPTION 1 RESPONSE
         DS    X                    . 'SET BUFFER ADDRESS' ORDER
         DS    X                    . SBA LOCATION 1
         DS    X                    . SBA LOCATION 2
         DS    X                    . OPTION 2 RESPONSE
         DS    X                    . 'SET BUFFER ADDRESS' ORDER
         DS    X                    . SBA LOCATION 1
         DS    X                    . SBA LOCATION 2
         DS    X                    . OPTION 3 RESPONSE
RDBUFLEN EQU   *-READBUFF           READ BUFFER LENGTH
*
TERM2@   DC    V(DKNTERM2)          GRAPHICS INTERFACE @
APPLBH@  DC    V(APPLBH)            APPLICATION B HELP ROUTINE @
APPLB1@  DC    V(APPLB1)            APPLICATION B OPTION 1 ROUTINE @
APPLB2@  DC    V(APPLB2)            APPLICATION B OPTION 2 ROUTINE @
APPLB3@  DC    V(APPLB3)            APPLICATION B OPTION 3 ROUTINE @
*
         END
```

Figure 1-6 (Part 4 of 4). DKNTERM2 Example 2

Determining Route Codes

For a CPCS-I application task to communicate with a CPCS-I user executive task, first it must know the user executive task's route code. This is done by calling DKNGETXI.

Linkage

The standard MVS/ESA (*) linkage conventions are followed.

The following interface control blocks are used to invoke DKNGETXI:

Table 1-6. DKNGETXI Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
User Executive Task Parameter List Extension • XTSKNAM (assembler) / EXEC-TASK-NAME (COBOL) must be loaded with the full 8-character name of the user executive task prior to calling DKNGETXI. • XTSKRETC (assembler) / EXEC-TASK-ROUTE-CODE (COBOL) contains the user executive task's route code on return from DKNGETXI.	EXINFOC	EXINFO

If XTSKRETC (assembler) / EXEC-TASK-ROUTE-CODE (COBOL) contains X'FF' (HIGH-VALUES) upon return from DKNGETXI, this means that DKNGETXI could not find the user executive task.

Restrictions

Do not refer to any User Executive Task Parameter List Extension field other than those listed above.

Example 1

Operation:

Determine the route code for a user executive task named DKNJANE. Send two messages to it.

Note: See "Sending Messages to User Executive Tasks" on page 1-27 for information on module DKNGETB2.

COBOL Code:

```
     ID DIVISION.
     PROGRAM-ID. TSGETXI.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
    /
     WORKING-STORAGE SECTION.

     01  WS-USEX-AREA.
         05  WS-USEX-NAME.
             10  FILLER               PIC  X(03) VALUE 'DKN'.
             10  WS-USEX-TASK-NAME    PIC  X(04) VALUE 'JANE'.
             10  FILLER               PIC  X(01) VALUE SPACE.
         05  WS-USEX-ROUTE-CODE       PIC  X(01).
             88  WS-NO-ROUTE-CODE-FOUND          VALUE HIGH-VALUE.

     01  WS-DKNGETB2-AREA.
         05  WS-DKNGETB2-ROUTE-CODE   PIC  X(01).
             88  WS-DKNGETB2-ALL-OK              VALUE LOW-VALUE.
         05  WS-DKNGETB2-TASK-NAME    PIC  X(04).
         05  WS-DKNGETB2-MESSAGE      PIC  X(56).

     01  WS-MESSAGE-AREA.
         05  WS-VERY-IMPORTANT-MSG    PIC  X(56) VALUE
             'SEE SPOT RUN.'.
         05  WS-EQUALLY-IMPORTANT-MSG PIC  X(56) VALUE
             'RUN, SPOT, RUN.'.

     COPY EXINFOC.

         ...................
         ...................
         ...................

    /
     LINKAGE SECTION.

     COPY DKNCATCB.

     01  START-PARMS                  PIC   X(68).
    /
     PROCEDURE DIVISION               USING APTCB
                                            START-PARMS.

         PERFORM 1000-FIND-ROUTE-CODE.

         ...................
         ...................
         ...................
```

Figure 1-7 (Part 1 of 2). DKNGETXI Example 1

```
        MOVE WS-VERY-IMPORTANT-MSG   TO WS-DKNGETB2-MESSAGE.
        PERFORM 2000-SEND-MESSAGE.

        MOVE WS-EQUALLY-IMPORTANT-MSG
                                     TO WS-DKNGETB2-MESSAGE.
        PERFORM 2000-SEND-MESSAGE.

        ...................
        ...................
        ...................

        GOBACK.

    1000-FIND-ROUTE-CODE.

        MOVE WS-USEX-NAME            TO EXEC-TASK-NAME.

        CALL 'DKNGETXI'              USING APTCB,
                                         EXECUTIVE-TASK-WORK-AREA.

        MOVE EXEC-TASK-ROUTE-CODE    TO WS-USEX-ROUTE-CODE.

        IF WS-NO-ROUTE-CODE-FOUND
            PERFORM 9999-SUFFER-PANIC-ATTACK
        END-IF.


    2000-SEND-MESSAGE.

        MOVE WS-USEX-ROUTE-CODE      TO WS-DKNGETB2-ROUTE-CODE.
        MOVE WS-USEX-TASK-NAME       TO WS-DKNGETB2-TASK-NAME.

        CALL 'DKNGETB2'              USING APTCB,
                                           WS-DKNGETB2-AREA.

        IF WS-DKNGETB2-ALL-OK
            CONTINUE
        ELSE
            PERFORM 9999-SUFFER-PANIC-ATTACK
        ENDIF.

        ...................
        ...................
        ...................

    9999-SUFFER-PANIC-ATTACK.

        ...................
        ...................
        ...................
```

Figure 1-7 (Part 2 of 2). DKNGETXI Example 1

Example 2

Operation:

Determine the route code for a user executive task named OXYMRON. Send two messages to it.

Note: See “Sending Messages to User Executive Tasks” on page 1-27 for information on module DKNGETB2.

Assembler Code:

```
TSGETXI  CSECT
TSGETXI  AMODE 31
TSGETXI  RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB@
*
         LOAD  EPLOC=GETXINME,      LOAD DKNGETXI               X
               ERRET=PANIC          ON FAILURE PROCESS ERROR
         ST    R0,GETXI@            SAVE DKNGETXI @
*
         LOAD  EPLOC=GETB2NME,      LOAD DKNGETB2               X
               ERRET=PANIC          ON FAILURE PROCESS ERROR
         ST    R0,GETB2@            SAVE DKNGETB2 @
*
         STORAGE OBTAIN,            OBTAIN EXINFO STORAGE       X
               LENGTH=XTSKLEN,                                  X
               ADDR=(R9),                                       X
               SP=8,                                            X
               COND=YES
         LTR   R15,R15              SUCCESSFUL OPERATION?
         BNZ   PANIC                N. PROCESS ERROR
*
         USING EXINFO,R9            ESTABLISH ADDRESSABILITY
         MVC   XTSKNAM,USEXNAME     SUPPLY USER EXEC NAME
         MVC   PARM1,APTCB@         SET APTCB @
         ST    R9,PARM2             SET EXINFO @
         LA    R1,PARMS             GET DKNGETXI PARM LIST @
         L     R15,GETXI@           GET DKNGETXI @
         BASSM R14,R15              CALL DKNGETXI
```

Figure 1-8 (Part 1 of 3). DKNGETXI Example 2

```
*
         MVC   USEXRTC,XTSKRETC       SAVE USEX ROUTE CODE
         CLI   USEXRTC,NOROUTE        OPERATION FAILED?
         BE    PANIC                  Y. PROCESS ERROR.
*
         ...................
         ...................
         ...................
*
         MVC   GETB2RTC,USEXRTC       SUPPLY USEX ROUTE CODE
         MVC   GETB2TSK,USEXTASK      SUPPLY USEX TASK NAME
         MVC   GETB2MSG,MESSAGE       SUPPLY MESSAGE
         MVC   PARM1,APTCB@           SET APTCB @
         LA    R1,GETB2BFR            GET DKNGETB2 BUFFER @
         ST    R1,PARM2               SET DKNGETB2 BUFFER @
         LA    R1,PARMS               GET DKNGETB2 PARM LIST @
         L     R15,GETB2@             GET DKNGETB2 @
         BASSM R14,R15                CALL DKNGETB2
*
         CLI   GETB2RTC,GETB2OK       SUCCESSFUL OPERATION?
         BNE   PANIC                  N. PROCESS ERROR
*
         ...................
         ...................
         ...................
*
         B     RETURN                 ALL DONE, EXIT
*
PANIC    DS    0H                     ERROR ROUTINE
         LA    R8,8                   SET BAD RETURN CODE
         B     RETURNX                CLEAN UP RESOURCES
*
RETURN   DS    0H                     GOOD EXIT
         XR    R8,R8                  SET GOOD RETURN CODE
*
RETURNX  DS    0H                     CLEAN UP RESOURCES
         DELETE EPLOC=GETXINME        RELEASE DKNGETXI
         DELETE EPLOC=GETB2NME        RELEASE DKNGETB2
*
         STORAGE RELEASE,             FREE EXINFO STORAGE         X
               LENGTH=XTSKLEN,                                     X
               ADDR=(R9),                                          X
               SP=8,                                               X
               COND=YES
*
         LR    R15,R8                 SET RETURN CODE
         L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
USEXNAME DS   0CL8                    USER EXECUTIVE TASK NAME
         DC    CL3'OXY'               . PREFIX
USEXTASK DC    CL4'MRON'              . TASK NAME
         DC    C' '                   . FILLER
```

Figure 1-8 (Part 2 of 3). DKNGETXI Example 2

```
*
USEXRTC  DS    XL1                    USEX TASK ROUTE CODE
NOROUTE  EQU   X'FF'                  USEX NOT FOUND RETURN CODE
*
GETB2BFR DS   0CL61                   DKNGETB2 MESSAGE BUFFER
GETB2RTC DS    XL1                    . ROUTE CODE / RETURN CODE
GETB2OK  EQU   X'00'                  . DKNGETB2 SUCCESSFUL
GETB2TSK DS    CL4                    . TASK NAME
GETB2MSG DS    CL56                   . MESSAGE
*
MESSAGE  DC    CL56'A COMPETENT PROGRAMER.'    MESSAGE
*
GETXINME DC    CL8'DKNGETXI'          DKNGETXI MODULE NAME
GETB2NME DC    CL8'DKNGETB2'          DKNGETB2 MODULE NAME
*
APTCB@   DS    A                      APTCB    @
GETXI@   DS    A                      DKNGETXI @
GETB2@   DS    A                      DKNGETB2 @
*
PARMS    DS   0A                      FUNCTION CALL PARM LIST
PARM1    DS    A                      . 1ST PARM
PARM2    DS    A                      . 2ND PARM
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
         COPY  EXINFO                 EXTENSION LIST DSECT
*
         END
```

Figure 1-8 (Part 3 of 3). DKNGETXI Example 2

Sending Messages to User Executive Tasks

If a CPCS-I application task knows the route code for a CPCS-I user executive task, it can send messages to it. This is done by calling DKNGETB2.

Linkage

The standard MVS/ESA (*) linkage conventions are followed.

The following interface control blocks are used to invoke DKNGETB2:

Table 1-7. DKNGETB2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
Communication Buffer	EXINFOC	EXINFO

- 1 byte route code, as returned from a DKNGETXI call.
- 60 bytes free-form message.

Although the message is free-form, it is recommended that its first four bytes be the task name of the user executive task. Messages can also be set to a user executive task from a terminal; these messages will always contain the task name in the first four bytes.

Example 1

Operation:

Determine the route code for a user executive task named DKNJANE. Send two messages to it.

Note: See “Determining Route Codes” on page 1-21 for information on module DKNGETXI.

COBOL Code:

```
     ID DIVISION.
     PROGRAM-ID. TSGETB2.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
```

Figure 1-9 (Part 1 of 4). DKNGETB2 Example 1

```
      /
       WORKING-STORAGE SECTION.

       01  WS-USEX-AREA.
           05  WS-USEX-NAME.
               10  FILLER               PIC  X(03) VALUE 'DKN'.
               10  WS-USEX-TASK-NAME    PIC  X(04) VALUE 'JANE'.
               10  FILLER               PIC  X(01) VALUE SPACE.
           05  WS-USEX-ROUTE-CODE       PIC  X(01).
               88  WS-NO-ROUTE-CODE-FOUND          VALUE HIGH-VALUE.

       01  WS-DKNGETB2-AREA.
           05  WS-DKNGETB2-ROUTE-CODE   PIC  X(01).
               88  WS-DKNGETB2-ALL-OK              VALUE LOW-VALUE.
           05  WS-DKNGETB2-TASK-NAME    PIC  X(04).
           05  WS-DKNGETB2-MESSAGE      PIC  X(56).

       01  WS-MESSAGE-AREA.
           05  WS-VERY-IMPORTANT-MSG    PIC  X(56) VALUE
               'SEE SPOT RUN.'.
           05  WS-EQUALLY-IMPORTANT-MSG PIC  X(56) VALUE
               'RUN, SPOT, RUN.'.

       COPY EXINFOC.

           ...................
           ...................
           ...................

      /
       LINKAGE SECTION.

       COPY DKNCATCB.

       01  START-PARMS                  PIC   X(68).
      /
       PROCEDURE DIVISION               USING APTCB
                                              START-PARMS.

           PERFORM 1000-FIND-ROUTE-CODE.

           ...................
           ...................
           ...................
```

Figure 1-9 (Part 2 of 4). DKNGETB2 Example 1

```
        MOVE WS-VERY-IMPORTANT-MSG   TO WS-DKNGETB2-MESSAGE.
        PERFORM 2000-SEND-MESSAGE.

        MOVE WS-EQUALLY-IMPORTANT-MSG
                                     TO WS-DKNGETB2-MESSAGE.
        PERFORM 2000-SEND-MESSAGE.

        ...................
        ...................
        ...................

        GOBACK.

    1000-FIND-ROUTE-CODE.

        MOVE WS-USEX-NAME            TO EXEC-TASK-NAME.

        CALL 'DKNGETXI'              USING APTCB,
                                         EXECUTIVE-TASK-WORK-AREA.

        MOVE EXEC-TASK-ROUTE-CODE    TO WS-USEX-ROUTE-CODE.

        IF WS-NO-ROUTE-CODE-FOUND
            PERFORM 9999-SUFFER-PANIC-ATTACK
        END-IF.

    2000-SEND-MESSAGE.

        MOVE WS-USEX-ROUTE-CODE      TO WS-DKNGETB2-ROUTE-CODE.
        MOVE WS-USEX-TASK-NAME       TO WS-DKNGETB2-TASK-NAME.

        CALL 'DKNGETB2'              USING APTCB,
                                           WS-DKNGETB2-AREA.

        IF WS-DKNGETB2-ALL-OK
            CONTINUE
        ELSE
            PERFORM 9999-SUFFER-PANIC-ATTACK
        ENDIF.

        ...................
        ...................
        ...................
```

Figure 1-9 (Part 3 of 4). DKNGETB2 Example 1

```
    9999-SUFFER-PANIC-ATTACK.

        ...................
        ...................
        ...................
```

Figure 1-9 (Part 4 of 4). DKNGETB2 Example 1

Example 2

Operation:

Determine the route code for a user executive task named OXYMRON. Send two messages to it.

Note: See "Determining Route Codes" on page 1-21 for information on module DKNGETXI.

Assembler Code:

```
TSGETB2  CSECT
TSGETB2  AMODE 31
TSGETB2  RMODE ANY
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)          GET APTCB@
*
         LOAD  EPLOC=GETXINME,       LOAD DKNGETXI               X
               ERRET=PANIC           ON FAILURE PROCESS ERROR
         ST    R0,GETXI@             SAVE DKNGETXI @
*
         LOAD  EPLOC=GETB2NME,       LOAD DKNGETB2               X
               ERRET=PANIC           ON FAILURE PROCESS ERROR
         ST    R0,GETB2@             SAVE DKNGETB2 @
```

Figure 1-10 (Part 1 of 3). DKNGETB2 Example 2

```
*
         STORAGE OBTAIN,               OBTAIN EXINFO STORAGE       X
               LENGTH=XTSKLEN,                                     X
               ADDR=(R9),                                          X
               SP=8,                                               X
               COND=YES
         LTR   R15,R15                 SUCCESSFUL OPERATION?
         BNZ   PANIC                   N. PROCESS ERROR
*
         USING EXINFO,R9               ESTABLISH ADDRESSABILITY
         MVC   XTSKNAM,USEXNAME        SUPPLY USER EXEC NAME
         MVC   PARM1,APTCB@            SET APTCB @
         ST    R9,PARM2                SET EXINFO @
         LA    R1,PARMS                GET DKNGETXI PARM LIST @
         L     R15,GETXI@              GET DKNGETXI @
         BASSM R14,R15                 CALL DKNGETXI
*
         MVC   USEXRTC,XTSKRETC        SAVE USEX ROUTE CODE
         CLI   USEXRTC,NOROUTE         OPERATION FAILED?
         BE    PANIC                   Y. PROCESS ERROR.
*
         ...................
         ...................
         ...................
*
         MVC   GETB2RTC,USEXRTC        SUPPLY USEX ROUTE CODE
         MVC   GETB2TSK,USEXTASK       SUPPLY USEX TASK NAME
         MVC   GETB2MSG,MESSAGE        SUPPLY MESSAGE
         MVC   PARM1,APTCB@            SET APTCB @
         LA    R1,GETB2BFR             GET DKNGETB2 BUFFER @
         ST    R1,PARM2                SET DKNGETB2 BUFFER @
         LA    R1,PARMS                GET DKNGETB2 PARM LIST @
         L     R15,GETB2@              GET DKNGETB2 @
         BASSM R14,R15                 CALL DKNGETB2
*
         CLI   GETB2RTC,GETB2OK        SUCCESSFUL OPERATION?
         BNE   PANIC                   N. PROCESS ERROR
*
         ...................
         ...................
         ...................
*
         B     RETURN                  ALL DONE, EXIT
*
PANIC    DS    0H                      ERROR ROUTINE
         LA    R8,8                    SET BAD RETURN CODE
         B     RETURNX                 CLEAN UP RESOURCES
*
RETURN   DS    0H                      GOOD EXIT
         XR    R8,R8                   SET GOOD RETURN CODE
*
RETURNX  DS    0H                      CLEAN UP RESOURCES
         DELETE EPLOC=GETXINME         RELEASE DKNGETXI
         DELETE EPLOC=GETB2NME         RELEASE DKNGETB2
```

Figure 1-10 (Part 2 of 3). DKNGETB2 Example 2

```
*
         STORAGE RELEASE,              FREE EXINFO STORAGE         X
               LENGTH=XTSKLEN,                                     X
               ADDR=(R9),                                          X
               SP=8,                                               X
               COND=YES
*
         LR    R15,R8                  SET RETURN CODE
         L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
USEXNAME DS   0CL8                     USER EXECUTIVE TASK NAME
         DC    CL3'OXY'                . PREFIX
USEXTASK DC    CL4'MRON'               . TASK NAME
         DC    C' '                    . FILLER
*
USEXRTC  DS    XL1                     USEX TASK ROUTE CODE
NOROUTE  EQU   X'FF'                   USEX NOT FOUND RETURN CODE
*
GETB2BFR DS   0CL61                    DKNGETB2 MESSAGE BUFFER
GETB2RTC DS    XL1                     . ROUTE CODE / RETURN CODE
GETB2OK  EQU   X'00'                   . DKNGETB2 SUCCESSFUL
GETB2TSK DS    CL4                     . TASK NAME
GETB2MSG DS    CL56                    . MESSAGE
*
MESSAGE  DC    CL56'A COMPETENT PROGRAMMER.'    MESSAGE
*
GETXINME DC    CL8'DKNGETXI'           DKNGETXI MODULE NAME
GETB2NME DC    CL8'DKNGETB2'           DKNGETB2 MODULE NAME
*
APTCB@   DS    A                       APTCB    @
GETXI@   DS    A                       DKNGETXI @
GETB2@   DS    A                       DKNGETB2 @
*
PARMS    DS   0A                       FUNCTION CALL PARM LIST
PARM1    DS    A                       . 1ST PARM
PARM2    DS    A                       . 2ND PARM
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
         COPY  EXINFO                  EXTENSION LIST DSECT
*
         END
```

Figure 1-10 (Part 3 of 3). DKNGETB2 Example 2

Writing User Executive Tasks

A CPCS-I user executive task is an executive task that users write. It is designed to run in the background, providing a variety of services and functions that can be called upon by sending the user executive task appropriate messages.

Restrictions:

- The user executive task's DKNBLDL entry must include the EXTSK=Y and MAX=1 parameters.
- The DKNBLDL entry can optionally include AUTO=1 and the EXSEQ= parameters. The former causes the user executive task to automatically start when CPCS-I comes up (the task does not have to be started by an operator). The latter controls the order in which multiple user executive tasks start.
- The DKNBLDL entry can include the INSTRG= and OUTSTRG= parameters, if the task conducts Mass Data Set I/O. All other DKNBLDL parameters are ignored.
- User executive tasks may call the CPCS-I service routines DKNMASS, DKNGETB2, DKNCYCI, and DKNPCTLI. No other CPCS-I programs or functions should be invoked.

The user executive task must be written as follows:

Step 1

Upon startup, the user executive task receives as a parameter list pointers to the CPCS-I Parameter List, and to the user executive task's APTCB. The following code fragment can be used to establish addressability:

Assembler Code:

```
         LR    R2,R1                  LOAD ETSKPARM ADDRESS
         USING ETSKPARM,R2            ADDRESSABILITY
*
         L     R11,ETSKPRM1           LOAD PARMLST ADDRESS
         USING PARMLST,R11            ADDRESSABILITY
*
         L     R6,SAVAPTCB            LOAD APTCB ADDRESS
         USING APTCB,R6               ADDRESSABILITY
         DROP  R2                     NO LONGER NEED ETSKPARM
*
         ..................
         ..................
         ..................
```

Figure 1-11 (Part 1 of 2). Setting Up Addressability for User Executive Task

```
*
         COPY  EXTSKPARM              USER EXEC START-UP PARM LIST
         EJECT
         COPY  DKNPARM                CPCS-I PARM LIST
         EJECT
         COPY  DKNAPTCB               OUR APTCB
         EJECT
```

Figure 1-11 (Part 2 of 2). Setting Up Addressability for User Executive Task

Step 2

The user executive task should search through the parameter list extension to find its extension block, communication buffer code, and communications ECB. The following code fragment demonstrates how:

Assembler Code:

```
         L     R12,ETSKEXTA           GET EXTENSION ADDRESS
         USING XTSKNAM,R12            ADDRESSABILITY
*
NEXTNAME CLC   XTSKNAM,XFF            Q/END OF TABLE
         BE    PANIC                  A/YES, WE ARE NOT HERE (THIS
*                                     SHOULD NOT HAPPEN)
         CLC   XTSKNAM,USEXNAME       Q/IS THIS US
         BE    GOTMYNAM               A/YES, GO TO MAIN PROCESS
         LA    R12,XTSKLEN(,R12)      POINT TO NEXT ENTRY
         B     NEXTNAME               LOOP TO SCAN THROUGH TABLE
*
GOTMYNAM DS    0H                     FOUND OUR EXTENSION
*
         ...................
         ...................
         ...................
*
PANIC    DS    0H                     DID NOT FIND OUR EXTENSION
         L     R13,4(R13)             GET CALLER'S SAVE AREA ADDRESS
         DKNAMODE TYPE=RETURN,RC=8    RETURN WITH ERROR
*
         ...................
         ...................
         ...................
```

Figure 1-12 (Part 1 of 2). User Executive Task Searching Parameter List Extension

```
*
XFF      DC    8X'FF'                EXTENSION LIST TERMINATOR
USEXNAME DC    CL8'DKNJANE '         NAME OF THIS USER EXEC TASK
         EJECT
         COPY  EXINFO                EXTENSION BLOCK
         EJECT
```

Figure 1-12 (Part 2 of 2). User Executive Task Searching Parameter List Extension

If a user executive task cannot find its name in one of the parameter list extension fields, it should end.

Note: Although the CPCS-I service routine DKNGETXI also scans the parameter extension list, it returns a copy of the found extension block, rather than a pointer to it. The communications ECB XTSKECBC as returned by DKNGETXI is unusable, because it is a copy of an ECB and not the real thing.

Step 3

The user executive task should next construct a communication buffer TRT table, with which it can interrogate DKNBMGR for inbound messages. The following code fragment shows how:

Assembler Code:

```
         XR    R15,R15               CLEAR WORK REGISTER
         IC    R15,XTSKRTEC          GET OUR DESTINATION CODE
         STC   R15,MYTRT(R15)        SET IT IN THE TRANS. TABLE
*
         ...................
         ...................
         ...................
*
MYTRT    DC    XL256'00'
         ORG   MYTRT+9
         DC    X'FF'
         ORG   ,
```

Figure 1-13. User Executive Task Constructing Communication Buffer TRT Table

The extra flag at address MYTRT+9 is required.

Step 4

The user executive task should wait for inbound communication buffers by issuing a WAIT, as follows:

Assembler Code:

```
BIGWAIT  DS    0H
         WAIT  1,ECB=XTSKECBC          WAIT FOR COMM. BUFFERS
         L     R4,XTSKECBC             GET POST CODE IN THE COMM ECB
         XC    XTSKECBC,XTSKECBC       CLEAR COMM ECB FOR NEXT POST
         SLL   R4,2                    STRIP POST/WAIT BITS
         SRL   R4,2                    ..
         C     R4,SHUTFLAG             Q/IS THIS A SHUTDOWN POST
         BNE   SCANBUFF                A/NO, CONTINUE TO PROCESS
         L     R13,4(R13)              GET CALLER'S SAVE AREA ADDRESS
         DKNAMODE TYPE=RETURN,RC=0     RETURN
*
         ...................
         ...................
         ...................
*
         DS    0F
SHUTFLAG DC    X'00FFFFFF'             "CPCS-I IS SHUTTING DOWN" FLAG
```

Figure 1-14. User Executive Task Wait For Inbound Communication Buffers

Step 5

The user executive task should scan for inbound communication buffers by calling DKNBMGR at its +4 EPA. The following code fragment shows how:

Assembler Code:

```
 SCANBUFF DS    0H
          LA    R0,MYTRT                TRTABLE
          L     R1,BUFSTAT              GET PNTR TO BUFFER STAT BYTES
          LR    R2,R11                  PARMLST POINTER
          L     R15,ABUFMGR             BUFFER MANAGER ADDRESS
          BAS   R14,4(R15)              CALL HIM
```

Figure 1-15 (Part 1 of 2). User Executive Task Scan For Inbound Communication Buffers

```
*
*        AT THIS POINT,
*
*          R0 = ADDRESS OF FOUND COMMUNICATION BUFFER (IF ANY)
*
*          R1 = ADDRESS OF THE BUFFER STATUS BYTE.
*             = 0 IF NO COMMUNICATION BUFFER FOUND.
*
*          R2 = BUFFER STATUS BYTE IN LOW-ORDER BYTE; THE THREE
*               HIGH-ORDER BYTES SHOULD BE IGNORED.
*
         LTR   R1,R1                   Q/DID WE FIND A BUFFER
         BZ    BIGWAIT                 N/GO FOR MORE WORK
```

Figure 1-15 (Part 2 of 2). User Executive Task Scan For Inbound Communication Buffers

Step 6

The user executive task should unload the communications buffer and free it as soon as possible. The following code fragment shows how:

Assembler Code:

```
FOUNDBUF DS    0H
         LR    R5,R0                  COPY ADDRESS OF THE BUFFER
         USING DKNCBUFF,R5            ADDRESSABILITY
         MVC   COMMBSVE(L'CBUFOLEN),COMMBUFF  SAVE COMM BUFFER DATA
         LA    R8,COMMBSVE            POINT TO NEW BUFFER AREA
         DROP  R5                     ADDRESSABILITY
         USING CBUFFREE,R8            ADDRESSABILITY
         MVI   0(R1),0                FREE ORIGINAL BUFFER TO CPCS
*
         ...................
         ...................
         ...................
*
COMMBSVE DS    CL(L'CBUFOLEN)         SAVED COMM BUFFER COPY
*
         COPY  DKNCBUFF
```

Figure 1-16. User Executive Task Freeing Communication Buffers

Step 7

The user executive task should now analyze the saved buffer contents and perform whatever functions or services have been requested of it.

When it is finished, the user executive task should return to label SCANBUFF, in case other communication buffers are waiting for it. If none are, then it should WAIT on XTSKECBC (label BIGWAIT) until more arrive. The user executive task should never end unless a message arrives explicitly asking it to.

Example

Operation:

Write a user executive task that accepts four commands:

CMD1 Display a message indicating that this command was received.

CMD2 Display a message indicating that this command was received.

STOP Causes the user executive task to shut down.

DUMP Causes the user executive task to abend with a dump.

Display messages as the program runs documenting its behavior. All messages are displayed to the terminal originating the command, to the supervisor terminal, and on the JES log.

Note: Member DKNEXT1 in CPCSI.V01R01.SDKNSAM2 contains the assembler code for this example.

Issuing Messages

CPCS-I application tasks may process messages calling module DKNSMSG, which allows you to:

- Retrieve messages for use by the calling task.
- Display messages on the system, MICR or INSC supervisor terminals.
- Send messages as WTOs to the MVS/ESA system console.
- Send messages to the scroll log.
- Send messages to the CPCS-I log (APTR ddname).
- Any combination of the above.

Also, COBOL applications can write WTO messages to the system operator by calling the module DKNWTO. See “Example 3” on page 1-45.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNSMSG:

Table 1-8 (Page 1 of 2). DKNSMSG Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
Program Name • This area contains the 3-character subsystem identifier, followed by the task name without its first three characters.	DKNCRMSG	DKNSMSGP
Message Number • This area contains the message severity level followed by the 4-character message number.	DKNCRMSG	DKNSMSGP
Return Code • This area contains the DKNSMSG return code in decimal.	DKNCRMSG	DKNSMSGP
Destination • This area identifies the message destination(s). Valid IDs are: system, MICR and INSC supervisor terminal, scroll log, MVS/ESA console, ATASK log and calling program.	DKNCRMSG	DKNSMSGP
Buffer • This area contains messages returned to the caller.	DKNCRMSG	DKNSMSGP

Table 1-8 (Page 2 of 2). DKNSMSG Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Insert String • This area contains a character string to be inserted in the message, followed by a LOW-VALUE/binary zeroes byte. An insert string control block is required for each character string inserted in the message.		

DKNSMSG may return the following completion codes in the return code control block:

Table 1-9. DKNSMSG Return Codes

Code (char)	Description
00	Successful completion.
01	Invalid destination ID.
02	No communications buffer available.
03	Dynamic module load failure.
04	Dynamic storage allocation failure.
05	VSAM access error. Scroll log messages clarify the problem.
06	Invalid APTCB address.
07	Invalid output buffer address.
08	Truncated returned/issued message.
09	Invalid insert character-string count.
0A	Message number not in CPCS-I message data set.
0B	Overflowed DKNSMSG internal buffer.
0C	Invalid format message.
0D	DKNSMSG internal error.
0E	Invalid number of parameters. This may be caused by the last parameter not having its highest order bit on (assembler programs).
0F	Null parameter address passed.
10	DKNSMSG not initialized.
11	The CPCS-I message data set cannot be opened because the CPCS-I VSAM table does not contain the data set's ID.

DKNSMSG also returns the completion code in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs).

Restrictions

Messages must be defined in the CPCS-I message data set. See the *CPCS-I Customization Guide* for documentation on how to customize the CPCS-I message data set.

Generic messages get the subsystem and task names inserted in their prefix area.

Example 1

Operation:

Perform the following tasks:

1. Retrieve generic message `TEST 39001 STRING EEEE-P-HH-HH-HH-HH-T-SSS OPEN ERR RC=X`, which shows the string name and DKNMASS module return code on string open failures, from the message data set.
2. Display user message `TEST 00003 INVALID START PARAMETER EEEE`, where EEEE is the task start parameter, on the system supervisor terminal. It is assumed that the message is defined in the message data set with a severity level of 0 for the task TEST.
3. Display user message `TEST 00006 TASK TERMINATED` on the MVS/ESA console and write it to the scroll log. It is assumed that the message is defined in the message data set with a severity level of 0 for the task TEST.

COBOL Code:

```
    ID DIVISION.
    PROGRAM-ID. TSSMSG.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.

    01  TASK-NAME                  PIC  X(05) VALUE 'TEST'.

    01  OPEN-STRING-NAME.
        05 FILLER                  PIC  X(24).
        05 FILLER                  PIC  X(01) VALUE LOW-VALUE.

    01  DKNMASS-RETURN-CODE.
        05 FILLER                  PIC  X(03) VALUE 'RC='.
        05 DR-CODE                 PIC  X(01).
        05 FILLER                  PIC  X(01) VALUE LOW-VALUE.

    01  MESSAGE-NUMBERS.
        05  MN-OPEN-ERROR          PIC  9(05) VALUE 39001.
        05  MN-INVALID-PARMS       PIC  9(05) VALUE 00003.
        05  MN-TASK-TERMINATED     PIC  9(05) VALUE 00006.

    COPY DKNCRMSG.

    01  INSERT-1                   PIC  X(25).
    01  INSERT-2                   PIC  X(06).
    01  INSERT-3                   PIC  X(05).
   /
    LINKAGE SECTION.

    COPY DKNCATCB.
```

Figure 1-17 (Part 1 of 3). DKNSMSG Example 1

```
 01   START-PARMS.
      05  SP-ENTRY-NUMBER       PIC  X(04).
      05  FILLER                PIC  X(64).
/
 PROCEDURE DIVISION                  USING APTCB
                                         START-PARMS.

     ...................
     ...................
     ...................

     PERFORM 1000-RETRIEVE-MESSAGE-1.

     ...................
     ...................
     ...................

     PERFORM 2000-SEND-MESSAGE-2.

     ...................
     ...................
     ...................

     PERFORM 3000-SEND-MESSAGE-3.

     ...................
     ...................
     ...................

     GOBACK.

 1000-RETRIEVE-MESSAGE-1.

     MOVE ' '                        TO SMSG-SUB-SYSTEM.
     MOVE TASK-NAME                  TO SMSG-PROGRAM.
     MOVE MN-OPEN-ERROR              TO SMSG-NUMBER.
     MOVE SMSG-DEST-RETURN-MSG       TO SMSG-DESTINATION.
     MOVE OPEN-STRING-NAME           TO INSERT-1.
     MOVE DKNMASS-RETURN-CODE        TO INSERT-2.

     CALL 'DKNSMSG'                  USING APTCB
                                         SMSG-PROGRAM-NAME
                                         SMSG-NUMBER
                                         SMSG-RETURN-CODE
                                         SMSG-DESTINATION
                                         SMSG-BUFFER
                                         INSERT-1
                                         INSERT-2.

 2000-SEND-MESSAGE-2.
```

Figure 1-17 (Part 2 of 3). DKNSMSG Example 1

```
        MOVE ' '                        TO SMSG-SUB-SYSTEM.
        MOVE TASK-NAME                  TO SMSG-PROGRAM.
        MOVE MN-INVALID-PARMS           TO SMSG-NUMBER.
        MOVE SMSG-DEST-SUPV-TERM        TO SMSG-DESTINATION.
        MOVE SP-ENTRY-NUMBER            TO INSERT-3.

        CALL 'DKNSMSG'                  USING APTCB
                                              SMSG-PROGRAM-NAME
                                              SMSG-NUMBER
                                              SMSG-RETURN-CODE
                                              SMSG-DESTINATION
                                              SMSG-BUFFER
                                              INSERT-3.

    3000-SEND-MESSAGE-3.

        MOVE ' '                        TO SMSG-SUB-SYSTEM.
        MOVE TASK-NAME                  TO SMSG-PROGRAM.
        MOVE MN-TASK-TERMINATED         TO SMSG-NUMBER.
        COMPUTE SMSG-DESTINATION        =  SMSG-DEST-WTO
                                        +  SMSG-DEST-SCROLL-DS.

        CALL 'DKNSMSG'                  USING APTCB
                                              SMSG-PROGRAM-NAME
                                              SMSG-NUMBER
                                              SMSG-RETURN-CODE
                                              SMSG-DESTINATION
                                                 SMSG-BUFFER.
```

Figure 1-17 (Part 3 of 3). DKNSMSG Example 1

Example 2

Operation:

Create a CPCS-I task called TEST, under the subsystem SYS, that:

1. Displays generic message `SYSTEST 19041 INVALID CYCLE NUMBER ENTERED` on the INSC supervisor termimal.
2. Displays user message `SYSTEST 30002 TEST COMPLETE; XXXXXX RECORDS PROCESSED` on the system supervisor terminal. It is assumed that the message is defined in the message data set with a severity level of 3, and will therefore also be sent to the calling program and the scroll log.

Assembler Code:

```
TSSMSG   CSECT
TSSMSG   AMODE 31
TSSMSG   RMODE ANY
*
```

Figure 1-18 (Part 1 of 2). DKNSMSG Example 2

```
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         MVC   APTCB1@,0(R1)          GET APTCB @
         MVC   APTCB2@,0(R1)          GET APTCB @
*
         LOAD  EPLOC=SMSGNAME,ERRET=LOADERR LOAD MESSAGE HANDLER
         ST    R0,SMSG@               SAVE MESSAGE HANDLER @
         .................
         .................
         .................
*
         MVC   SMPSUBS,SYSSUB         SET SUBSYSTEM NAME
         MVC   SMPPROG,TESTTSK        SET TASK NAME
         MVC   SMPNUM,MSG1NUM         SET MESSAGE NUMBER
         LA    R4,SMPDINSC            SET INSC SUPERVISOR
         STH   R4,SMPDEST               DESTINATION
         OI    RETBUFF1,ENDLIST       SET 'END OF LIST' FLAG
         LA    R1,PARMS1              GET MESSAGE HANDLER PARMS @
         L     R15,SMSG@              GET MESSAGE HANDLER @
         BASSM R14,R15                CALL MESSAGE HANDLER
         CLC   SMPRC,SMSGRC00         SUCCESSFUL COMPLETION ?
         BNE   SMSGERR                N. PROCESS ERROR
         .................
         .................
         .................
*
         MVC   SMPSUBS,SYSSUB         SET SUBSYSTEM NAME
         MVC   SMPPROG,TESTTSK        SET TASK NAME
         MVC   SMPNUM,MSG2NUM         SET MESSAGE NUMBER
         LA    R4,SMPDSUPV            SET SYST SUPERVISOR
         STH   R4,SMPDEST               DESTINATION
         OI    INSRT12@,ENDLIST       SET 'END OF LIST' FLAG
         LA    R1,PARMS2              GET MESSAGE HANDLER PARMS @
         L     R15,SMSG@              GET MESSAGE HANDLER @
         BASSM R14,R15                CALL MESSAGE HANDLER
         CLC   SMPRC,SMSGRC00         SUCCESSFUL COMPLETION ?
         BNE   SMSGERR                N. PROCESS ERROR
         .................
         .................
         .................
*
SMSGERR  DS   0H                      PROCESS DKNSMSG ERROR
         .................
         .................
         .................
*
```

Figure 1-18 (Part 2 of 2). DKNSMSG Example 2

```
LOADERR  DS   0H                       PROCESS LOAD ERROR
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
PARMS1   DS   0F                       DKNSMSG PARM LIST
APTCB1@  DS    A                       . APTCB @
         DC    A(SMPPNAME)             . SUBSYSTEM & TASK NAME @
         DC    A(SMPNUM)               . MESSAGE NUMER @
         DC    A(SMPRC)                . RETURN CODE @
         DC    A(SMPDEST)              . DESTINATION @
RETBUFF1 DC    A(SMPBUF)               . RETURN BUFFER @
*
PARMS2   DS   0F                       DKNSMSG PARM LIST
APTCB2@  DS    A                       . APTCB @
         DC    A(SMPPNAME)             . SUBSYSTEM & TASK NAME @
         DC    A(SMPNUM)               . MESSAGE NUMER @
         DC    A(SMPRC)                . RETURN CODE @
         DC    A(SMPDEST)              . DESTINATION @
         DC    A(SMPBUF)               . RETURN BUFFER @
INSRT12@ DC    A(INSERT1)              . INSERT CHARACTER STRING 1 @
*
ENDLIST  EQU   X'80'                   'END OF LIST' FLAG
*
SMSGNAME DC    CL8'DKNSMSG'            MESSAGE HANDLER MODULE NAME
SYSSUB   DC    C'SYS'                  SUBSYSTEM NAME
TESTTSK  DC    C'TEST'                 TASK NAME
MSG1NUM  DC    C'19041'                MESSAGE 1 NUMBER
MSG2NUM  DC    C'30002'                MESSAGE 2 NUMBER
INSERT1  DS    CL6                     ITEM COUNT
SMSGRC00 DC    C'00'                   SMSG SUCCESSFUL COMPLETION RC
*
         DKNSMSGP BUFSIZE=618          DKNSMSG PARMS
*
SMSG@    DS    A                       MESSAGE HANDLER @
*
         END
```

Figure 1-18 (Part 3 of 2). DKNSMSG Example 2

Example 3

Operation:

Send a message to the system operator from a COBOL program.

The *message-name* consists of 4 bytes followed by the actual message. *message-name* requires the following:

- The first field of 2 bytes must contain the length of *message-name* (including the 4 bytes) in binary.
- The next 2 bytes should be binary zeros.

- The maximum message length is 121, so that the maximum value in the first 2-byte field is 125 in binary format.

COBOL Code:

```
    CALL 'DKNWTO' USING MESSAGE-NAME.
```

Figure 1-19. DKNWTO COBOL Example

Performing String I/O

CPCS-I applications may call module DKNMASS to perform string I/O. DKNMASS allows you to:

- Search for types of strings.
- Read string directories.
- Update string directories.
- Read strings.
- Create strings.
- Delete strings.

Valid search criteria are:

- All strings for a specified cycle ID.
- All M-strings for a specified cycle ID.
- All I-strings for a specified cycle ID.
- All D-strings for a specified cycle ID.
- All R-strings for a specified cycle ID.
- All W-strings for a specified cycle ID.
- All R-strings.
- All on-us D-strings.
- All subsequent pass reject D-strings.
- All kill D-strings for a specified endpoint ID.

Strings may be read sequentially or directly. Direct reads allow access to previously read codeline records whose note information was saved. Each successful ‘read’ returns a codeline record in both compressed and uncompressed formats.

Codeline records may be written in either compressed or uncompressed format.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNMASS:

Table 1-10. DKNMASS Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNMASS Call Parameters • The request code must always be specified. • The operation mode must be specified for 'end', 'open string', 'close string', 'read codeline record', 'write codeline record' and 'search directory' requests. The following table shows the different operation modes that may be specified for each request code.	DKNCRZA	ZADSCT
Expanded String Directory / Application Task Control Block • This control block is the APTCB only on 'initialization' calls. • This control block is not used on 'end' calls. • The cycle ID must be specified on output I-string and R-string 'open' calls. • The string name must be specified on 'open', 'read directory', 'write directory' and 'delete' calls. • The returned SSB Pointer, on 'open' calls, must be saved, and restored before subsequent 'read codeline'/'write codeline' record and 'close' requests. • The note information, returned on 'read directory' calls, must be restored before the subsequent 'write directory' call. The note should be 0 on the first 'search' / 'read sequential codeline record' call, and its value should be restored before subsequent 'search' / 'read sequential codeline record' calls.	DKNCRZE2/ DKNCATCB	ZE2DSCT/ DKNAPTCB
Compressed Codeline Record • This control block is only used on 'read codeline record' and 'write codeline record' calls.	DKNCRDI	DIDSCT
Uncompressed Codeline Record • This control block is only used on 'read codeline record' and 'write codeline record' calls. It is not required if only the compressed codeline record is used.	DKNCRZD	ZDDSCT

DKNMASS accepts the following operation modes in the call parameters:

Table 1-11. DKNMASS Operation Modes

Request Code	Operation Mode	Description
End	A	Purge request - This function closes open input strings, purges open output M-strings and D-strings, and puts in restart mode open output I-strings and R-strings. This function is used when the application task ends before completing its string I/O process.
	All others	Normal end - DKNMASS causes a U094 abend if any output string or any not completely processed input string has been left open.
Open String	1	Open string as input.
	2	Open restart I-string or R-string for further output.
	5	Open string as output.
Close String	K	Close an output kill D-string.
	C	Close an output on-us D-string or an output subsequent-pass reject
	0	Close any other type of string.
Read Codeline Record	1	Read sequentially.
	2	Read directly, using the note information in the string directory control block.
Search Directory	E	Search for strings whose cycle matches that in the string directory control block.
	M	Search for M-strings whose cycle matches that in the string directory control block.
	C	Search for R-strings, on-us D-strings, and subsequent pass reject D-strings, whose cycle matches that in the string directory control
	K	Search for kill D-strings whose endpoint matches that in the string directory control block.

DKNMASS may return the following completion codes in the call parameters communication control block:

Table 1-12 (Page 1 of 2). DKNMASS Return Codes

Code (hex)	Description
+00	Successful Completion.

Table 1-12 (Page 2 of 2). DKNMASS Return Codes

Code (hex)	Description
+01	'Open' request - one of the following occurred: • Too many strings open concurrently (MDEF MAXOPEN value may need to be increased) • Not enough main storage available (either CPCS-I job REGION or DKNBLDL OUTSTRG value may need to be increased) • There are not enough index records for output I-strings or R-strings (the index data set size must be increased, the MDEF directory entries must be re-specified, and DKNMTASK must be re-gened). 'Close' request - There are not enough index records for output M-strings or D-strings (the index data set size must be increased, the MDEF directory entries must be re-specified, and DKNMTASK must be re-gened).
+02	'Open' request - The output string already exists.
+03	'Open' request - The input string does not exist or is not complete.
+04	'Close' request - The string is not open. 'Read codeline record' request - The string is not open.
+05	'Read codeline record' request - End of string reached; String has been closed.
+06	'Read codeline record' request - Incorrect note value, or end of incomplete string that was opened in restart mode. 'Read directory' request - String directory cannot be located. 'Delete' request - The string cannot be located.
+07	'Open' request - The string has already been opened by this task. 'Delete' request - The string is open, and cannot be deleted.
+08	I/O error.
+09	'Search directory' request - No strings found.
+5E	'End' request - Open output strings (also a U094 abend occurs).
+FF	Invalid parameters.

Restrictions

Application tasks must always start by issuing a DKNMASS 'initialization' call before calling DKNMASS to perform any function, and must issue a DKNMASS 'END' call after completing their string I/O functions.

See "Describing an Application Task's Environment" in the *CPCS-I Customization Guide*, for a description of the DKNBLDL INSTRG and OUTSTRG parameters.

Important: A successful 'read directory' function must be followed by a 'write directory' function. Also, a 'write directory' function must follow a successful 'read directory' function. Failure to do this results in the eventual inability to process codeline records.

Warning: A 'read directory' operation issues exclusive control over the CPCS-I string indexing function, and no other task may use the function until a subsequent 'write directory' releases exclusive control. If it is necessary to do extensive processing between a 'read directory' and the associated 'write directory' to update a string's directory information, the following sequence of operations should be performed:

1. Read directory (and save furnished information).
2. Write directory (as is).
3. Perform extensive processing (on saved information).
4. Read directory.
5. Write directory (with updated information).

A string must be opened as 'input' before it can be read. An input string must only be closed if it is not read until the 'end of string' completion code is returned.

If an open output string needs to be read, it must first be closed and then opened as input.

A string must be opened as 'output' before records can be written into it. An output string must be closed after writing its last codeline record.

When a string is deleted via DKNMASS, its space is freed, but the associated pass-to-pass control information is not reset. See "Deleting Strings" on page 1-60 for a more effective method of deleting strings.

Tasks performing multiple string I/O, issuing a single mass data set services initialization, and using a single ZA control block, *must* save the SSB pointer after each open and *must* restore it before any subsequent DKNMASS call for the string. The pointer can be found in the ZE control block (in Cobol, the SSB pointer is in the ZE-SSB-PTR field; in Assembler, it is the ZESSBA field).

Example 1

Operation:

Create string 0004-1-88-00-00-00-M-000, copying into it all user code 88 codeline records from string 0004-1-00-00-00-00-M-000, with the same string flags and in the same cycle.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSMASS.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  INPUT-STRING-NAME.
     05  FILLER                          PIC  X(04) VALUE '0004'.
     05  FILLER                          PIC  X(01) VALUE '1'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(01) VALUE 'M'.
     05  FILLER                          PIC  X(03) VALUE '000'.

 01  OUTPUT-STRING-NAME.
     05  FILLER                          PIC  X(04) VALUE '0004'.
     05  FILLER                          PIC  X(01) VALUE '1'.
     05  FILLER                          PIC  X(02) VALUE '88'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(02) VALUE '00'.
     05  FILLER                          PIC  X(01) VALUE 'M'.
     05  FILLER                          PIC  X(03) VALUE '000'.

 01  RECORD-TYPE-88                      PIC  X(01) VALUE X'88'.
 01  INPUT-STG-SSB-PTR                   PIC S9(09)       COMP.
 01  OUTPUT-STG-SSB-PTR                  PIC S9(09)       COMP.

 COPY DKNCRZA.

 COPY DKNCRZE2                          REPLACING ==:ZE:==
                                        BY        ==ZE==.
 COPY DKNCRDI                           REPLACING ==:DI:==
                                        BY        ==DI==.

 COPY DKNCRZD                           REPLACING ==:ZD:==
                                        BY        ==ZD==.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01   START-PARMS                       PIC  X(68).
/
 PROCEDURE DIVISION                     USING APTCB
                                              START-PARMS.

     PERFORM 1000-INITIALIZE-DKNMASS.
```

Figure 1-20 (Part 1 of 3). DKNMASS Example 1

```
       IF  ZA-NO-ERROR
           PERFORM 2000-OPEN-STRINGS

           IF  ZA-NO-ERROR
               PERFORM 3000-PROCESS-STRING
                   UNTIL NOT ZA-NO-ERROR
               IF  ZA-END-OF-STG
                   PERFORM 4000-CLOSE-OUTPUT-STRING.

       PERFORM 5000-END-DKNMASS.

       GOBACK.

   1000-INITIALIZE-DKNMASS.

       SET  ZA-MDS-INIT                    TO TRUE.

       CALL 'DKNMASS'                      USING ZA
                                                 APTCB.

   2000-OPEN-STRINGS.

       PERFORM 2500-OPEN-INPUT-STRING.

       IF  ZA-NO-ERROR
           PERFORM 2700-OPEN-OUTPUT-STRING.

   2500-OPEN-INPUT-STRING.

       SET  ZA-OPEN-STG
            ZA-MODE-OPEN-INPUT             TO TRUE.
       MOVE INPUT-STRING-NAME              TO ZE-STG-NAME.

       CALL 'DKNMASS'                      USING ZA
                                                 ZE.

       IF  ZA-NO-ERROR
           MOVE ZE-SSB-PTR                 TO INPUT-STG-SSB-PTR.

   2700-OPEN-OUTPUT-STRING.

       SET  ZA-OPEN-STG
            ZA-MODE-OPEN-OUTPUT            TO TRUE.
       MOVE OUTPUT-STRING-NAME             TO ZE-STG-NAME.

       CALL 'DKNMASS'                      USING ZA
                                                 ZE.

       IF  ZA-NO-ERROR
           MOVE ZE-SSB-PTR                 TO OUTPUT-STG-SSB-PTR.

   3000-PROCESS-STRING.

       PERFORM 3500-READ-INPUT-STRING.
```

Figure 1-20 (Part 2 of 3). DKNMASS Example 1

```
    IF  ZA-NO-ERROR
        IF  ZD-USER-FLAGS              = RECORD-TYPE-88
            PERFORM 3700-WRITE-OUTPUT-STRING.

3500-READ-INPUT-STRING.

    SET  ZA-READ
         ZA-MODE-READ-SEQ              TO TRUE.
    MOVE INPUT-STG-SSB-PTR             TO ZE-SSB-PTR.

    CALL 'DKNMASS'                     USING ZA
                                             ZE
                                             DI
                                             ZD.

3700-WRITE-OUTPUT-STRING.

    SET  ZA-WRITE                      TO TRUE.
    MOVE OUTPUT-STG-SSB-PTR            TO ZE-SSB-PTR.

    CALL 'DKNMASS'                     USING ZA
                                             ZE
                                             DI
                                             ZD.

4000-CLOSE-OUTPUT-STRING.

    SET  ZA-CLOSE-STG                  TO TRUE.
    MOVE OUTPUT-STG-SSB-PTR            TO ZE-SSB-PTR.

    CALL 'DKNMASS'                     USING ZA
                                             ZE.

5000-END-DKNMASS.

    MOVE ZA-RETURN-CODE                TO RETURN-CODE.
    SET  ZA-MDS-END                    TO TRUE.

    IF  ZA-NO-ERROR
        SET  ZA-MODE-UNUSED            TO TRUE
    ELSE
        SET  ZA-MODE-END-PURGE         TO TRUE.

    CALL 'DKNMASS'                     USING ZA
                                             APTCB.
```

Figure 1-20 (Part 3 of 3). DKNMASS Example 1

Example 2

Operation:

Turn the ‘string transferred’ flag on in all cycle D M-strings.

Assembler Code:

```
TSMASS   CSECT
TSMASS   AMODE 24
TSMASS   RMODE 24
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         LA    R4,ZAINIT            SET 'DKNMASS START'
         STCM  R4,B'0011',ZAENTRY     REQUEST CODE
         MVC   PARM2,0(R1)          GET APTCB @
         OI    PARM2,ENDLIST        SET 'END OF LIST' FLAG
         LA    R1,PARMS             GET DKNMASS PARM LIST @
         L     R15,MASS@            GET DKNMASS @
         BASSM R14,R15              CALL DKNMASS TO INITIALIZE
         CLI   ZARETCD,ZANOERR      SUCCESSFUL OPERATION ?
         BNE   MASSERR              N. PROCESS DKNMASS ERROR
*
NEXTSTG  LA    R4,ZE                GET
         ST    R4,PARM2               ZE @
         MVC   ZENOTE,SVNOTE        GO THROUGH ALL STRINGS
         LA    R4,ZASDIR            SET 'SEARCH FOR STRING'
         STCM  R4,B'0011',ZAENTRY     REQUEST CODE
         MVI   ZAOPMOD,OPMSTG       SET STRING TYPE OPERATION MODE
         MVC   ZECYCLE,CYCLID       SET CYCLE ID
         OI    PARM2,ENDLIST        SET 'END OF LIST' FLAG
         LA    R1,PARMS             GET DKNMASS PARM LIST @
         L     R15,MASS@            GET DKNMASS @
         BASSM R14,R15              CALL DKNMASS TO SEARCH
         CLI   ZARETCD,ZAEDIR       MORE STRINGS ?
         BE    ENDMASS              N. END DKNMASS & EXIT PROGRAM
         CLI   ZARETCD,ZANOERR      SUCCESSFUL OPERATION ?
         BNE   MASSERR              N. PROCESS DKNMASS ERROR
*
         MVC   SVNOTE,ZENOTE        SAVE SEARCH NOTE FOR NEXT SEARCH
         LA    R4,ZARDIR            SET 'READ STG DIRECTORY'
         STCM  R4,B'0011',ZAENTRY     REQUEST CODE
         LA    R1,PARMS             GET DKNMASS PARM LIST @
         L     R15,MASS@            GET DKNMASS @
         BASSM R14,R15              CALL DKNMASS TO READ DIRECTORY
         CLI   ZARETCD,ZANOERR      SUCCESSFUL OPERATION ?
         BNE   MASSERR              N. PROCESS DKNMASS ERROR
*
```

Figure 1-21 (Part 1 of 2). DKNMASS Example 2

```
         MVI   ZEXFR,FLAGON          TURN TRANSFER FLAG ON
         LA    R4,ZAWDIR             SET 'WRITE STG DIRECTORY'
         STCM  R4,B'0011',ZAENTRY      REQUEST CODE
         LA    R1,PARMS              GET DKNMASS PARM LIST @
         L     R15,MASS@             GET DKNMASS @
         BASSM R14,R15               CALL DKNMASS TO UPDATE DIRECTORY
         CLI   ZARETCD,ZANOERR       SUCCESSFUL OPERATION ?
         BNE   MASSERR               N. PROCESS DKNMASS ERROR
         B     NEXTSTG               Y. PROCESS NEXT STRING
*
ENDMASS  LA    R4,ZAEND              SET 'END DKNMASS'
         STCM  R4,B'0011',ZAENTRY      REQUEST CODE
         MVI   ZAOPMOD,OPNORML       SET 'NORMAL END' OPERATION MODE
         OI    PARM1,ENDLIST         SET 'END OF LIST' FLAG
         LA    R1,PARMS              GET DKNMASS PARM LIST @
         L     R15,MASS@             GET DKNMASS @
         BASSM R14,R15               CALL DKNMASS TO END
         CLI   ZARETCD,ZANOERR       SUCCESSFUL OPERATION ?
         BE    RETURN                Y. EXIT PROGRAM
*
MASSERR  XR    R15,R15               CLEAR RETURN CODE REGISTER
         ICM   R15,B'0001',ZARETCD   PASS DKNMASS RETURN CODE
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
OPMODES  DS   0H                     DKNMASS OPERATION MODES
OPMSTG   EQU   C'M'                  . SEARCH FOR M-STRINGS
OPNORML  EQU   C' '                  . NORMAL END
*
SVNOTE   DC    D'0'                  SEARCH NOTE SAVE AREA
*
CYCLID   DC    CL2'D'                CYCLE ID
*
ENDLIST  EQU   X'80'                 'END OF LIST' FLAG
FLAGON   EQU   C'1'                  'ON' FLAG
*
MASS@    DC    V(DKNMASS)            DKNMASS @
*
PARMS    DS   0F                     DKNMASS PARM LIST
PARM1    DC    A(ZA)                 . CALL PARMS @
PARM2    DS    A                     . APTCB @ / ZE @
*
         COPY  ZADSCT                DKNMASS CALL PARMS
*
         COPY  ZE2DSCT               DKNMASS DATA PARMS
*
SAVEAREA DS  18F                     OUR SAVE AREA
*
         END
```

Figure 1-21 (Part 2 of 2). DKNMASS Example 2

Processing User Areas with the User Area Manager

The CPCS-I User Area Manager is designed to isolate user data from CPCS-I control blocks and other user data. The User Area Manager uses “global user areas” to accomplish this task. A global user area is a collection of individual user areas that are self-defined logical records consisting of a three-character prefix, a one-byte length, and a free-form data area of the length specified in the one-byte length field. The User Area Manager is invoked by calling module DKNUAM.

In the following example:

```
Prefix=ABC, Length=8, Data="12345678"
```

Prefix	Length	Data (Free Format)
ABC	X'08'	12345678

Note: User area prefixes that begin with “I”, “X”, and “Z” are reserved strictly for use by IBM.

To use this facility, a CPCS-I program simply calls DKNUAM and DKNUAM does the rest:

- Manages access to the global area
- Handles extract, insert, update and delete requests

DKNUAM processes the individual user area request and gives control back to the calling program. If there is a problem, DKNUAM gives control back to the calling program with a return code detailing the nature of the error.

Activation

DKNUAM is activated during CPCS-I system initialization and remains active until CPCS-I system termination. Field UAM@, in the CPCS-I parameter list control block, contains the DKNUAM load address.

Linkage

The standard MVS/ESA linkage conventions are followed. The following interface control blocks are used to invoke DKNUAM:

Table 1-13. DKNUAM Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNUAM Call Parameters	DKNCUAM2	DKNCUAM1
User Area I/O Buffer		

For all requests, the DKNUAM function code must be specified in the User Area Manager interface control block. The list of function codes and their meanings are as follows:

Function Code	Description
X'01'	Extract an individual user area from a global user area
X'02'	Insert an individual user area into a global user area
X'03'	Update an individual user area in a global user area
X'04'	Delete an individual user area from a global user area

The following DKNUAM interface control block fields are listed with the the functions for which they are required:

Cobol Field	Assembler Field	Description
UM-FUNCTION-CODE	UMREQUST	Request function code. Required for *all* requests.
UM-USER-AREA-ID	UMEXARID	Individual user area ID. Required for *all* calls.
UM-GLOBAL-AREA-BUFFER-ADR	UMGUABF@	Global user area address. Required for *all* calls.
UM-USER-AREA-BUFFER-SIZE	UMBFSIZE	User area I/O buffer size. Required for EXTRACT requests. The individual user area gets truncated if the buffer size is less than the individual user area size.
UM-USER-AREA-SIZE	UMUASIZE	Individual user area size. Required for INSERT and UPDATE requests and is returned for *all* calls.

DKNUAM returns the following completion codes:

Table 1-14. DKNUAM Return Codes

Return Code	Description
0	Request processsed successfully
4	Invalid input parameters
8	User area not found
12	User area already present in global user area
16	No space available in global user area
20	Individual user area smaller than input area
24	Invalid input area length
28	Invalid individual user area length

Restrictions

The calling program is responsible for handling each of the DKNUAM exception conditions.

Example

```
CALLUAM  DKNPLINK FUNCT=SUBENTRY,SAVAREA=STACK
*
*-------------------------------------------------------------------*
*       Initialize the required User Area Manager Parameters        *
*-------------------------------------------------------------------*
*
         MVI   UMREQUST,UMINSERT       Request a user area Insert
         L     R1,EIGHT                Set size of eight (8) bytes
         STH   R1,UMUASIZE             Save for User Area Manager
         MVC   UMEXARID,=CL3'IBM'      Indicate prefix of IBM
         MVC   UMGUABF@,SDEUA@         Pass Global User Area Address
         LA    R1,UMINCNTL             Get Request Block Address
         ST    R1,UAMPARM1             Save for User Area Manager
         LA    R1,UAMDATA              Get Data Area Address
         ST    R1,UAMPARM2             Save for User Area Manager
*
*-------------------------------------------------------------------*
*                     Call the User Area Manager                    *
*-------------------------------------------------------------------*
*
         LA    R1,UAMPARMS             Get User Area Manager Parmlist @
         L     R15,UAM@                Get User Area Manager Address
         BASSM R14,R15                 Call the User Area Manager
         LTR   R15,R15                 Were we successful?
         BZ    CALLUA98                Yes...We are outta here...

                              .
                              .
                              .
                              .
                              .

*
*-------------------------------------------------------------------*
*                    User Area Manager Variables                    *
*-------------------------------------------------------------------*
*
         DKNCUAM1 DSECT=NO             UAM Request Block
*
UAMPARMS DS    0F                      UAM Parameter List
UAMPARM1 DC    A(0)                    UAM Request Block Address
UAMPARM2 DC    A(0)                    UAM Data Area Address
*
UAMDATA  DC    XL8'00'                 Time Stamp Save Area
```

Figure 1-22. DKNUAM Example

Deleting Strings

CPCS-I applications may call module DKNSTGD to delete strings more efficiently than through DKNMASS (see “Performing String I/O” on page 1-47). DKNSTGD not only deletes strings but also sends completion messages to the scroll log. Message DKNATASK LOG$, which includes the application name and the deleted string’s name, is sent after successful string deletions.

Linkage

Only the application task control block, containing the string’s name in the ‘start parameters’ field, is passed to DKNSTGD.

DKNSTGD may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 1-15. DKNSTGD Return Codes

Code (hex)	Description
+00	Successful Completion.
+04	String Not Found.
+10	Failed Deletion.

Error messages are sent to the scroll log when the DKNSTGD return code is not zero.

Restrictions

The calling program must also set the APTCB operator ID field to low-value (COBOL) or binary zeroes (assembler).

The calling program is responsible for saving the original contents of the modified APTCB fields before the DKNSTGD call, and restoring them after the DKNSTGD call.

Example 1

Operation:

Delete string 0004-1-00-00-00-00-I-000.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSSTGD.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  STRING-NAME.
     05  FILLER                        PIC X(04) VALUE '0004'.
     05  FILLER                        PIC X(01) VALUE '1'.
     05  FILLER                        PIC X(02) VALUE '00'.
     05  FILLER                        PIC X(02) VALUE '00'.
     05  FILLER                        PIC X(02) VALUE '00'.
     05  FILLER                        PIC X(02) VALUE '00'.
     05  FILLER                        PIC X(01) VALUE 'I'.
     05  FILLER                        PIC X(03) VALUE '000'.

 01  STGD-APTCB-TID                    PIC X(04).
 01  STGD-SPARM                        PIC X(56).
 01  SUCCESSFUL-OPERATION              PIC 9(01) VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                       PIC X(68).
/
 PROCEDURE DIVISION                    USING APTCB
                                             START-PARMS.

     ...................
     ...................
     ...................

     MOVE APTCB-TID                    TO STGD-APTCB-TID.
     SET  APTCB-AUTOMATIC-START        TO TRUE.
     MOVE APTCB-SPARM                  TO STGD-SPARM.
     MOVE STRING-NAME                  TO APTCB-SPARM.

     CALL 'DKNSTGD'                    USING APTCB.

     MOVE STGD-APTCB-TID               TO APTCB-TID.
     MOVE STGD-SPARM                   TO APTCB-SPARM.

     IF  RETURN-CODE                   =  SUCCESSFUL-OPERATION
         ...............
         ...............
         ...............

     GOBACK.
```

Figure 1-23. DKNSTGD COBOL Example 1

Assembler Code:

```
TSSTGD   CSECT
TSSTGD   AMODE 24
TSSTGD   RMODE 24
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         L     R4,0(R1)              GET APTCB @
         USING APTCB,R4              BASE APTCB
         .................
         .................
         .................
*
         MVC   SVTID,TCBTID          SAVE OPERATOR ID
         MVC   SVSPARM,TCBSPARM      SAVE START PARMS
         XC    TCBTID,TCBTID         CLEAR OPERATOR ID
         MVC   TCBSPARM(17),STGNAME  SET STRING NAME
         ST    R4,APTCB@             SET DKNSTGD PARM LIST
         LA    R1,APTCB@             GET DKNSTGD PARM LIST @
         L     R15,STGD@             GET DKNSTGD @
         BASSM R14,R15               CALL DKNSTGD
         MVC   TCBTID,SVTID          RESTORE OPERATOR ID
         MVC   TCBSPARM,SVSPARM      RESTORE START PARMS
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BNZ   RETURN                N. EXIT WITH DKNSTGD RET CODE
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
SAVEAREA DS  18F                     OUR SAVE AREA
*
STGNAME  DC    C'00041000000000I000' STRING NAME
*
APTCB@   DS    F                     DKNSTGD PARM LIST
*
SVTID    DS    F                     OPERATOR ID SAVE AREA
SVSPARM  DS    CL56                  START PARMS SAVE AREA
*
```

Figure 1-24 (Part 1 of 2). DKNSTGD Assembler Example 1

```
STGD@    DC    V(DKNSTGD)             DKNSTGD @
*
         COPY  DKNAPTCB               APTCB
*
         END
```

Figure 1-24 (Part 2 of 2). DKNSTGD Assembler Example 1

Accessing/Updating Pass-to-Pass Control Information

CPCS-I application tasks may call module DKNPCTLI to retrieve and update pass-to-pass control information. DKNPCTLI allows you to:

- Get the pass-to-pass information for a tracer ID.
- Get the pass-to-pass information for the next tracer group in a string.
- Get the pass-to-pass information for a pass pocket history and tracer group.
- Update the pass-to-pass information for a tracer ID.
- Delete the pass-to-pass information for a cycle.
- Determine whether pass-to-pass information exists for a tracer group.
- Create pass-to-pass information for a tracer ID.
- Delete the pass-to-pass information for a tracer group.
- Allocate a tracer group.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNPCTLI:

Table 1-16. DKNPCTLI Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNPCTLI Call Parameters • See next table for a description of the valid request codes.	DKNCRTL	TLDSCT
Pass-to-Pass Control Information	DKNCRTG	TGDSCT
Application Task Control Block	DKNCATCB	DKNAPTCB

DKNPCTLI accepts the following request codes:

Table 1-17 (Page 1 of 2). DKNPCTLI FUNCTIONS

Function	Description
Retrieve Tracer Data (RTD)	Gets tracer data for tracer ID. The tracer group number and slip number must be set in the pass-to-pass information control block before calling DKNPCTLI.
Retrieve Next Tracer Data (RNTD)	Gets tracer data for the next tracer group in the string. An RTD, RTDP or another RNTD must precede the request.
Retrieve Tracer Data Pass History (RTDP)	Gets tracer data for a pass pocket history and a tracer group. The tracer group number and pass pocket history must be set in the pass-to-pass information control block before calling DKNPCTLI.

Table 1-17 (Page 2 of 2). DKNPCTLI FUNCTIONS

Function	Description
Store Tracer Data (STD)	Updates existing pass-to-pass data by tracer ID. The tracer group number and slip number must be set in the pass-to-pass information control block before calling DKNPCTLI. The last entry and last tracer group ID fields must also contain correct information.
Delete Tracer Data (DTD)	Deletes pass-to-pass data for a cycle. The cycle ID must be set in the call parameters control block before calling DKNPCTLI.
Verify Tracer Group (VTG)	Determines whether there is pass-to-pass data for a tracer group. The tracer group number must be set in the pass-to-pass information control block before calling DKNPCTLI.
Add Tracer Data (ATD)	Creates pass-to-pass data for a tracer ID. The tracer group number and slip number must be set in the pass-to-pass information control block before calling DKNPCTLI.
Delete Tracer Data Group (DTDG)	Deletes all pass-to-pass data for a tracer group. The tracer group number must be set in the pass-to-pass information control block before calling DKNPCTLI.
Allocate new Tracer Group (ANTG)	Obtains a non-used tracer group number. Returned tracer group numbers start with 9999 and proceed downward. The tracer group number is returned in the pass-to-pass information control block. The tracer group maximum allowed number of slips is returned in the call parameters control block.

DKNPCTLI may return the following completion codes in the call parameters:

Table 1-18. DKNPCTLI Return Codes

Code (dec)	Description
+00	Successful completion.
+02	Invalid request.
+03	I/O error.
+04	Record not found.
+05	String's next tracer group does not exit (RNTD requests).
+06	Duplicate record.
+07	Data set full.

Restrictions

The communication control block reserved and work field contents must not be altered by the calling program.

Example 1

Operation:

Call DKNPCTLI to:

1. Determine whether tracer slip 011, in entry 0004, was sent to a rehandle pocket on prime pass, and if so, what pocket it was. It is assumed the I-string was distributed and no tracers were sent to kill pockets.
2. Delete the pass-to-pass control information for tracer group 0005.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSPCTLI.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  TRACER-GROUP-1                    PIC 9(01) VALUE 4.
 01  TRACER-GROUP-2                    PIC 9(01) VALUE 5.
 01  TRACER-SLIP                       PIC 9(02) VALUE 11.
 01  PRIME-PASS                        PIC 9(01) VALUE 1.
 01  REJECT-POCKET                     PIC X(02) VALUE X'FFFF'.

 01  REHANDLE-POCKET-SW                PIC X(01) VALUE 'N'.
     88  RP-REHANDLE-POCKET                      VALUE 'Y'.

 01  SAVE-REHANDLE-POCKET              PIC X(02).

 01  SUCCESSFUL-OPERATION              PIC 9(01) VALUE 0.

 COPY DKNCRTL.

 COPY DKNCRTG.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                       PIC X(68).
/
 PROCEDURE DIVISION                    USING APTCB
                                             START-PARMS.

     ...................
     ...................
     ...................

     PERFORM 1000-GET-TRACER-SLIP-DATA.
```

Figure 1-25 (Part 1 of 2). DKNPCTLI Example 1

```
      IF  TL-NO-ERROR
          PERFORM 2000-DELETE-TRACER-GROUP-DATA

          IF  TL-NO-ERROR
              MOVE SUCCESSFUL-OPERATION   TO RETURN-CODE

              ...................
              ...................
              ...................

          ELSE
              MOVE TL-RETURN-CODE         TO RETURN-CODE.

      GOBACK.

  1000-GET-TRACER-SLIP-DATA.

      MOVE TRACER-GROUP-1                 TO TG-NO.
      MOVE TRACER-SLIP                    TO TG-SEQ.
      SET  TL-RTD                         TO TRUE.

      CALL 'DKNPCTLI'                     USING TL
                                                TG
                                                APTCB.

      IF  TL-NO-ERROR
          IF  TG-PASS                     =  PRIME-PASS
              IF  TG-PKT-1                NOT = REJECT-POCKET
                  SET  RP-REHANDLE-POCKET TO TRUE
                  MOVE TG-PKT-1           TO SAVE-REHANDLE-POCKET.

  2000-DELETE-TRACER-GROUP-DATA.

                  MOVE TRACER-GROUP-2     TO TG-NO
                  SET  TL-DTDG            TO TRUE

                  CALL 'DKNPCTLI'         USING TL
                                                TG
                                                APTCB.
```

Figure 1-25 (Part 2 of 2). DKNPCTLI Example 1

Example 2

Operation:

Delete the pass-to-pass control information for cycle 8.

Assembler Code:

```
TSPCTLI  CSECT
TSPCTLI  AMODE 24
TSPCTLI  RMODE 24
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)          GET APTCB @
         ..............
         ..............
         ..............
*
         LA    R4,TGADTD             SET 'DELETE BY CYCLE'
         STCM  R4,B'0011',ENTRYCDE     REQUEST CODE
         MVC   TLCYCLE,CYCLEID       SET CYCLE ID
         OI    APTCB@,ENDLIST        SET 'END OF LIST' FLAG
         LA    R1,PARMS              GET DKNPCTLI PARM LIST @
         L     R15,PCTLI@            GET DKNPCTLI @
         BASSM R14,R15               CALL DKNPCTLI TO INITIALIZE
         CLI   TGARC,TGARC0          SUCCESSFUL OPERATION ?
         BNE   PCTLIERR              N. PROCESS DKNPCTLI ERROR
*
         LA    R15,0                 SET RC = 'SUCCESSFUL OPERATION'
         B     RETURN                EXIT PROGRAM
*
PCTLIERR DS   0H                     PROCESS DKNPCTLI ERROR
         XR    R15,R15               CLEAR RETURN CODE REGISTER
         ICM   R15,B'0001',TGARC     TRANSFER DKNPCTLI RETURN CODE
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
CYCLEID  DC    CL2'8'                CYCLE ID
*
ENDLIST  EQU   X'80'                 'END OF LIST' FLAG
*
PCTLI@   DC    V(DKNPCTLI)           DKNPCTLI @
*
PARMS    DS   0F                     DKNPCTLI PARM LIST
         DC    A(TLCB)               . CALL PARMS @
         DC    A(TGCB)               . PASS-TO-PASS DATA @
APTCB@   DS    A                     . APTCB @
*
TLCB     DS   0H                     DKNPCTLI CALL PARMS
         COPY  TLDSCT
*
```

Figure 1-26 (Part 1 of 2). DKNPCTLI Example 2

```
TGCB     DS   0H                    PASS-TO-PASS CONTROL DATA
         COPY  TGDSCT
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         END
```

Figure 1-26 (Part 2 of 2). DKNPCTLI Example 2

Using the Tracer Group Update Utility

CPCS-I application tasks may call module DKNTGUT to update and recover pass-to-pass control information for a string. DKNTGUT allows you to:

- Update the pass-to-pass control information for a tracer ID.
- Create pass-to-pass control information for a tracer ID.
- Allocate a tracer group.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNTGUT:

Table 1-19. DKNTGUT Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNTGUT Call Parameters • See next table for a description of the valid request codes.	DKNCRTGU	TGUTIL
Pass-to-Pass Control Information	DKNCRTG	TGDSCT
Application Task Control Block	DKNCATCB	DKNAPTCB

DKNTGUT accepts the following request codes:

Table 1-20. DKNTGUT FUNCTIONS

Function	Description
Initialize TGUT (TGINIT)	Sets up work areas. Establishes communication with DKNPCTL.
Process Tracer Records (TGPROC)	Processes tracer records. Formats tracer prefix records. Updates pass-to-pass control information. Allocates tracer group.
End Processing (TGEND)	For subsets, establishes next tracer number. Writes tracer prefix record. Ends communication with DKNPCTL.

DKNTGUT may return the following completion codes in the call parameters:

Table 1-21 (Page 1 of 2). DKNTGUT Return Codes

Code (dec)	Description
+00	Successful completion.
+01	Invalid command.
+02	GETMAIN failure.
+03	Tracer not found.

Table 1-21 (Page 2 of 2). DKNTGUT Return Codes

Code (dec)	Description
+04	Tracer error for subsets.
+05	Invalid subset chain.
+06	Entry not found.
+07	Entry not found for subsets.
+08	Entry already exists.
+09	Tracer verify error.
+10	Tracer not found.
+11	Rehandle not found.

Restrictions

When you use this utility, it is important that the logical sequence of tracer processing is still observed. For example, an attempt to update the tracer data set for the pass-2 I-string without having the correct tracer entries from the prime I-string, causes a tracer verification error.

Example 1

Operation:

Initialize, perform processing, and end TGUT.

Step 1

Do the following to fill in the appropriate fields in the DKNTGUT access area.

- Move `TGINIT` to the command. This command tells DKNTGUT to:
  - Initialize all values in the access area.
  - Verify that the entry tracer is valid for this pass.
- Move your module name to the calling module field. This field is informational only.
- Move your string name to the DKNTGUT string-name field.
- Move to the correct fields the cycle ID, the sort type, and the note information for the string.
- Specify any special process flags.

  Specify the tracer overwrite flag (TGUOWF). The valid values are **Y** and **N**. To bypass the checks and to determine whether the entry tracer already exists for prime pass, place a **Y** in this field.

Check the access-area return code to verify a successful TGINIT.

Note: After the TGINIT is complete, do not update the DKNTGUT access-area work area. DKNTGUT initializes many values in this area, so any changes made to this area after the TGINIT call might cause errors.

Step 2

Perform the process command.

- Move TGPROC to the DKNTGUT access-area command field.
- Call DKNTGUT, using the following conventions for each item in the string:

Check the access-area return code after each call to DKNTGUT.

Note: While performing the process command, do not update the DKNTGUT access-area work area. DKNTGUT initializes many values in this area, and any changes made to this area after the TGINIT call might cause errors.

Step 3

When the end of the string, perform the following step:

- Move TGEND to the DKNTGUT access-area command field.
- Call DKNTGUT:

Check the access-area return code after each call to DKNTGUT.

Assembler Code:

```
TGINT    DS    0H
         OC    SBFTGUTL,SBFTGUTL       TG AREA EXIST?
         BNZ   TGINT07                 YES..INIT FIELDS
         LA    R4,TGULEN               SAVE STRING BUFFER SIZE
*        GETMAIN RC,LV=(R4),LOC=ANY    GETMAIN STORAGE AREA
         GETMAIN RC,
               LV=(R4),
               LOC=BELOW,SP=XX
         LTR   R15,R15                 DID WE GET IT ?
         BNZ   ERROR                   NO...RETURN WITH ERROR
         ST    R1,SBFTGUTL             SAVE ADDR OF SBF AREA
*
TGINT07  DS    0H
         L     R3,SBFTGUTL             SET R3 TO TG AREA
         USING TGUTIL,R3               ESTABLISH ADDRESSABILITY
         MVC   TGUCMD,=C'TGINIT  '     PERFORM INIT COMMAND
         MVI   TGUOWF,TGUOWT           SET OVERWRITE FLAG ON
         BAL   R14,TGFIL
         BAL   R14,TGCAL                 CALL TG UTILITY
         B     RETURN                  RETURN FROM CALL
               .................
               .................
               .................
```

Figure 1-27 (Part 1 of 4). TGUT Assembler Example

```
*
TGUPD    DS    0H
         CLI   ZBSTGTYP,C'D'          IS THIS A D-STRING ?
         BE    TGUPD05                 YES..GO AHEAD
         CLI   ZBSTGTYP,C'I'          IS THIS A I-STRING ?
         BE    TGUPD05                 YES..GO AHEAD
         CLI   ZBSTGTYP,C'R'          SKIP UPDATE
         BNE   TGUPD990                NO...SO FORGET IT
*
TGUPD05  DS    0H
         L     R3,SBFTGUTL             SET R3 TO TG AREA
         LTR   R3,R3                   GOOD ADDR?
         BZ    TGUPD990                NO...SO FORGET IT
         MVC   TGUCMD,=C'TGPROC  '     PERFORM PROCESS COMMAND
         BAL   R14,TGFIL
         BAL   R14,TGCAL               CALL TG UTILITY
*
TGUPD990 DS    0H
         LM    R1,R8,TGREGS            RESTORE CALLERS REGISTERS
         B     RETURN                  AND RETURN TO CALLER
               .................
               .................
               .................
*
TGEND    DS    0H
         CLI   ZBSTGTYP,C'I'           I-STRING ?
         BNE   TGENDZ                  NO - SO FORGET IT
         L     R3,SBFTGUTL             SET R3 TO TG AREA
         LTR   R3,R3                   GOOD ADDR?
         BZ    TGENDZ                  NO...SO FORGET IT
         MVC   TGUCMD,=C'TGEND   '     PERFORM END     COMMAND
         BAL   R14,TGFIL
         BAL   R14,TGCAL               CALL TG UTILITY
         DELETE EP=DKNTGUT             DELETE LOADED TGUT
         XC    TGUTADDR,TGUTADDR       CLEAR OUT ADDRESS
         B     RETURN                  AND RETURN TO CALLER
               .................
               .................
               .................
```

Figure 1-27 (Part 2 of 4). TGUT Assembler Example

```
*
FREETG   DS    0H
         STM   R1,R8,TGREGS            GET SOME REGISTERS
FREETG05 DS    0H
         CLC   SBFTGUTL,=F'0'          ADDRESS LOWVALUES?
         BE    FREETG10                YES..NO FREEMAIN
         L     R2,SBFTGUTL             SET R2 TO AREA
         LA    R3,TGULEN               SAVE STRING BUFFER SIZE
*        FREEMAIN RC,A=(R2),LV=(R3)    AND FREE IT
         FREEMAIN RC,
               A=(R2),
               LV=(R3),SP=XX
         LTR   R15,R15                 GOOD FREE?
         BZ    RETURN                  YES..NO MESSAGE
         B     ERROR                   POINT TO ERROR MSG
               .................
               .................
               .................
*
TGFIL    DS    0H
         MVC   TGUMOD,=C'DKNXXXX'      MOVE CALLING MODULE NAME
         UNPK  TGUEEEE(5),ZBTRGP(3)    MOVE ENTRY & PASS
         MVC   TGUEEEE+4(1),ZBPASS     MOVE ENTRY & PASS
         MVC   TGUPKT1,ZBP1PKT         MOVE PKT
         CLC   =X'FFFF',ZBP1PKT        REJECT POCKET ?
         BNE   TGFIL10                 NO...LEAVE AS IS
         MVC   TGUPKT1,=C'R '          MOVE FIELD TO SCRN
*
TGFIL10  DS    0H
         MVC   TGUPKT2,ZBP2PKT         MOVE PKT
         CLC   =X'FFFF',ZBP2PKT        REJECT POCKET ?
         BNE   TGFIL20                 NO...LEAVE AS IS
         MVC   TGUPKT2,=C'R '          MOVE FIELD
*
TGFIL20  DS    0H
         MVC   TGUPKT3,ZBP3PKT         MOVE PKT
         CLC   =X'FFFF',ZBP3PKT        REJECT POCKET ?
         BNE   TGFIL30                 NO...LEAVE AS IS
         MVC   TGUPKT3,=C'R '          MOVE FIELD
*
TGFIL30  DS    0H
         MVC   TGUPKT4,ZBP4PKT         MOVE PKT
         CLC   =X'FFFF',ZBP4PKT        REJECT POCKET ?
         BNE   TGFIL40                 NO...LEAVE AS IS
         MVC   TGUPKT4,=C'R '          MOVE FIELD
*
TGFIL40  DS    0H
         MVC   TGUTYPE,ZBSTGTYP        MOVE STRING TYPE
         MVC   TGUSUB,WORKSUB          MOVE SUBSET NUMBER
         MVC   TGUCYCL,ZBCYCLE         MOVE CYCLE
         MVC   TGUSORT,ZBTYPEWK        SET SORT TYPE
         MVC   TGUNOTE,ZBNOTE          MOVE NOTE INFO
         BR    R14                     GO BACK
```

Figure 1-27 (Part 3 of 4). TGUT Assembler Example

```
*
TGCAL    DS    0H
         ST    R3,TGPACS               SAVE ACCESS AREA ADDR
         MVC   TGPAPTCB,APTCBAD        GET APTCB ADDRESS
         L     R1,MDSREC
         MVC   TGPDI,MDSREC            GET THE DI RECORD ADDRESS
         ICM   R15,15,TGUTADDR         GET TGUT ENTRY POINT ADDR
         BNZ   TGCAL10                 CONTINUE IF LOADED
         LOAD  EP=DKNTGUT
         ST    R0,TGUTADDR
         LR    R15,R0
*
TGCAL10  DS    0H
         LA    R1,TGPARMS              POINT R1 AT PARMLIST
         BASSM R14,R15                 EXECUTE USER ROUTINE
         L     R3,SBFTGUTL             SET R3 TO TG AREA
         CLC   TGURET,=C'0000'         GOOD RETURN?
         BNE   ERROR                   YES..RETURN TO CALLER
         B     RETURN                  YES..RETURN TO CALLER
               .................
               .................
               .................
*
SBFTGUTL DS    F            ADDRESS OF DKNTGUT WORK AREA
TGUTADDR DC    F'0'         ADDRESS OF DKNTGUT
         COPY  TGUTIL
         COPY  TGDSCT
         COPY  TGPARMS
```

Figure 1-27 (Part 4 of 4). TGUT Assembler Example

Example 2

Operation:

Initialize, perform processing, and end TGUT.

COBOL Code:

```
    CALL 'DKNTGUT' USING TGUTIL,
                         APTCB.
```

Figure 1-28. Calling TGUT COBOL Example

```
     CALL 'DKNTGUT' USING TGUTIL,
                          APTCB,
                          DI.
```

Figure 1-29. TGUT Processing COBOL Example

```
CALL 'DKNTGUT' USING TGUTIL,
                     APTCB.
```

Figure 1-30. Ending TGUT COBOL Example

Accessing/Updating Cycle Control Information

Application tasks may call module DKNCYCI to access or change the CPCS-I cycle control information. DKNCYCI may perform the following operations for any given cycle:

- Modify the cycle status.
- Modify the end-prime status.
- Obtain the current cycle and end-prime status.
- Obtain the current cycle and endorse dates.

Notes:

1. DKNCYCI calls the CYCL exit (see “CYCL Exit — Validate Cycle and Endorse Dates” on page 2-12) if active.
2. See “CYCL - Cycle Status”, in the *CPCS-I Terminal Operations Guide* for information on cycle status, end-prime status, cycle date and endorse date.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNCYCI:

Table 1-22 (Page 1 of 2). DKNCYCI Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB

Table 1-22 (Page 2 of 2). DKNCYCI Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNCYCI Call Parameters	DKNCCYC2	CYC2DSCT
B Activate cycle; not valid if the current cycle status is 'E'. Cycle and endorse dates are set to the current date if no dates are passed by the calling application. The format of the two dates must match the format specified by the DATE= parameter on the MDEF macro in the MTASK gen.		
D Deactivate cycle.		
E Cycle in end-cycle condition. Indicates that either DKNECYC or DKNECYC2 is processing the end-of-cycle activity. 'E' is valid only if the current status is 'D'; and when the processing is complete, it sets the status to back to 'D'.		
F Change end-prime status to 'deactivated'. Changes the end-prime status from 'A' to 'P' or from 'P' to 'D'. 'F' is not valid if the current end-prime status is 'D'.		
O Change end-prime status to 'activated'. Changes the end-prime status from 'D' to 'P' or from 'P' to 'A'. 'O' is not valid if the current end-prime status is 'A'.		
C Requests cycle date. Obtains the cycle and endorse dates, and the cycle and end-prime status.		

DKNCYCI may return the following return codes in both the call parameters and register 15:

Table 1-23. DKNCYCI Return Codes

Code (dec)	Description
+00	Successful completion.
+04	Invalid cycle ID.
+06	Invalid cycle status activation request (current status is E).
+07	Cycle status E change requested.
+08	End-prime status de-activation request for a current status of D.
+09	Invalid end-prime status change request.
+10	End-prime status activation request for a current status of A.
+12	Invalid cycle date.
+16	Cycle data set update error.
+20	Date validation error (see returned error message).

Example 1

Operation:

Obtain the current cycle and endorse dates and cycle and end-prime status for cycle 8. Activate end-prime if the current end-prime status is ‘P’.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSCYCI.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  CYCLE-ID                              PIC X(01)  VALUE '8'.

 COPY DKNCCYC2.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                           PIC X(68).
/
 PROCEDURE DIVISION                        USING APTCB
                                                 START-PARMS.

     ...................
     ...................
     ...................

     MOVE CYCLE-ID                         TO CYCI-CYCLE-ID.
     SET CYCI-REQUEST-DATE                 TO TRUE.

     CALL 'DKNCYCI'                        USING APTCB
                                                 DKNCYCI-PARMS.

     IF  CYCI-RETURN-GOOD
         IF  CYCI-END-PRIME-PENDING
             SET CYC2-REQUEST-ACT-END-PRIME TO TRUE

             CALL 'DKNCYCI'                USING APTCB
                                                 DKNCYCI-PARMS.

     MOVE CYCI-RETURN-CODE                 TO RETURN-CODE.

     ...................
     ...................
     ...................

     GOBACK.
```

Figure 1-31 (Part 1 of 2). DKNCYCI Example 1

Example 2

Operation:

Activate cycle 9.

Assembler Code:

```
TSCYCI   CSECT
TSCYCI   AMODE 31
TSCYCI   RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
         .................
         .................
         .................
*
         MVC   CYCICYCL,CYCLEID     SET CYCLE ID
         MVI   CYCIRQST,CYCREQB     SET 'ACTIVATE CYCLE' REQUEST CODE
         LA    R1,PARMS             GET DKNCYCI PARM LIST @
         L     R15,CYCI@            GET DKNCYCI @
         BASSM R14,R15              CALL DKNCYCI TO ACTIVATE CYCLE
         ICM   R15,B'1111',CYCIRETN PROPAGATE DKNCYCI RETURN CODE
         .................
         .................
         .................
*
         L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
CYCI@    DC    V(DKNCYCI)           DKNCYCI @
*
CYCLEID  DC    C'9'                 CYCLE ID
*
PARMS    DS   0F                    DKNCYCI PARM LIST
APTCB@   DS    A                   . APTCB @
         DC    A(CYCICYCL)         . CALL PARMS @
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         COPY  CYC2DSCT             DKNCYCI CALL PARMS
*
         END
```

Figure 1-32 (Part 1 of 2). DKNCYCI Example 2

Accessing Bank Control Information

CPCS-I applications may call module DKNBCFIO to obtain bank control information.

See “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for information on the Bank Control Data Set.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNBCFIO:

Table 1-24. DKNBCFIO Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Bank Number • This 3 digit field contains the target bank number. 000 is not a valid bank number.		
Returned Bank Control Information • Some of this information may not exist for the target bank, in which case it is obtained from the default bank data.	DKNCRBCF	BCF

DKNBCFIO returns a completion code in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs). See “DKNPDSIO — Perform PDS I/O Processing” on page 4-17 for a description of the return codes.

Example 1

Operation:

Obtain the name and address information for bank 001.

COBOL Code:

```
  ID DIVISION.
  PROGRAM-ID. TSBCFIO.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  BANK-NUMBER                          PIC X(03)  VALUE '001'.
```

Figure 1-33 (Part 1 of 2). DKNBCFIO COBOL Example 1

```
 01  NAME-AND-ADDRESS.
     05  NA-1                          PIC X(26).
     05  NA-2                          PIC X(26).
     05  NA-3                          PIC X(26).
     05  NA-4                          PIC X(26).

 01  SUCCESSFUL-OPERATION              PIC 9(01)  VALUE 0.

 COPY DKNCRBCF.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                       PIC X(68).
/
 PROCEDURE DIVISION                    USING APTCB
                                             START-PARMS.

     ...................
     ...................
     ...................

     CALL 'DKNBCFIO'                   USING BANK-NUMBER
                                             BCF-REC.

     IF  RETURN-CODE                   =  SUCCESSFUL-OPERATION
         MOVE BCF-NAMEADDR1            TO NA-1
         MOVE BCF-NAMEADDR2            TO NA-2
         MOVE BCF-NAMEADDR3            TO NA-3
         MOVE BCF-NAMEADDR4            TO NA-4.

     ...................
     ...................
     ...................

     GOBACK.
```

Figure 1-33 (Part 2 of 2). DKNBCFIO COBOL Example 1

Assembler Code:

```
TSBCFIO  CSECT
TSBCFIO  AMODE 24
TSBCFIO  RMODE 24
*
         SAVE  (14,12)                 SAVE REGISTERS
         BASR  R12,0                   GET ROUTINE @
         USING *,R12                   BASE SUBROUTINE
*
```

Figure 1-34 (Part 1 of 2). DKNBCFIO Assembler Example 1

```
         LA    R14,SAVEAREA            POINT TO OUR SAVE AREA
         ST    R14,8(R13)              STORE FORWARD SAVE AREA @
         ST    R13,4(R14)              STORE BACKWARD SAVE AREA @
         LR    R13,R14                 POINT TO NEW SAVE AREA
*
         ................
         ................
         ................
*
         LA    R1,PARMS                GET DKNBCFIO PARM LIST @
         L     R15,BCFIO@              GET DKNBCFIO @
         BASSM R14,R15                 CALL DKNBCFIO
         LTR   R15,R15                 SUCCESSFUL OPERATION ?
         BNZ   BCFIOERR                N. PROCESS DKNBCFIO ERROR
         MVC   NAMADDR1,BCFNADD1       GET NAME AND ADDRESS LINE 1
         MVC   NAMADDR2,BCFNADD2       GET NAME AND ADDRESS LINE 2
         MVC   NAMADDR3,BCFNADD3       GET NAME AND ADDRESS LINE 3
         MVC   NAMADDR4,BCFNADD4       GET NAME AND ADDRESS LINE 4
         ................
         ................
         ................
         B     RETURN                  EXIT PROGRAM
*
BCFIOERR DS   0H                       PROCESS DKNBCFIO ERROR
         ................
         ................
         ................
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
BCFIO@   DC    V(DKNBCFIO)             DKNBCFIO @
*
PARMS    DS   0F                       DKNBCFIO PARM LIST
         DS    A(BANKNBR)              . BANK NUMBER FIELD @
         DC    A(BCFREC)               . RETURN AREA @
*
BANKNBR  DC    C'001'                  TARGET BANK NUMBER
*
NAMADDR  DS   0H                       BANK NAME AND ADDRESS
NAMADDR1 DS    CL26                    . LINE 1
NAMADDR2 DS    CL26                    . LINE 2
NAMADDR3 DS    CL26                    . LINE 3
NAMADDR4 DS    CL26                    . LINE 4
*
         COPY  BCF                     DKNBCFIO RETURN AREA
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
         END
```

Figure 1-34 (Part 2 of 2). DKNBCFIO Assembler Example 1

Finding the Next Available Kill-Bundle Record

Application programs that need to find the next available kill-bundle record, or need to know how many records are in the kill-bundle file, should use DKNLINK2. DKNLINK2 is normally called by COBOL applications, but may be called by assembler applications.

When a program needs only to find the RBA of the next available block, it can place a nonblank character in the eighth position of `FILE-NAME` or `FILEINFO`. In this case, DKNKEY does not increment the RBA.

Linkage

The standard MVS/ESA linkage conventions are followed.

There are no copybooks for this function.

Restrictions

Do not run DKNLINK2 while DKNCOMP is active.

Example 1

Operation:

Find the next available kill-bundle record.

COBOL Code:

```
 01  KEY-PARM.
     03  FILE-NAME               PIC X(8)       VALUE 'DKNKB  X'.
     03  KB-NOMINAL-KEY          PIC S9(8) COMP VALUE +0.
     03  KEY-ERROR-CODE          PIC X          VALUE '0'.
 01  LINK-RETURN-CODE            PIC S9(8) COMP VALUE ZERO.
     ..................
     ..................
     ..................
/
 LINKAGE SECTION.
     COPY DKNCATCB.
     ..................
     ..................
     ..................
/
     CALL 'DKNLINK2' USING KEY-PARM LINK-RETURN-CODE APTCB.
     IF  LINK-RETURN-CODE NOT EQUAL ZERO
     THEN
         GO TO LINK-ERROR.
     MOVE KB-NOMINAL-KEY TO KB-NOMINAL-KEY-SAVE.
     MOVE -1 TO KB-NOMINAL-KEY.
```

Figure 1-35. DKNLINK2 COBOL Example

Example 2

Operation:

An assembler program needs to find the next available kill-bundle record.

Assembler application programs can issue the following call:

Assembler Code:

```
CALL DKNLINK2,(FILEINFO,LRETCODE,(Rx)),VL
```

Figure 1-36. DKNLINK2 Assembler Example

Where `Rx` is a register containing the address of the APTCB and `LRETCODE` is the address of the return code from DKNLINK or DKNLINK2.

Obtaining Unique Sequence Numbers

CPCS-I application tasks may call module DKNIGEN to obtain unique codeline record sequence numbers. A contiguous set of sequence numbers may be reserved for the application on a single call to DKNIGEN.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNIGEN:

Table 1-25. DKNIGEN Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
DKNIGEN Call parameters	DKNCRIGN	DKNIGPRM

- The request code must always be set to 'I'.
  - The logical document processor number must always be set to 0.
  - The volume field must be set to one if DKNIGEN is called for each processed codeline record, or to the number of sequence numbers to be reserved for the application if DKNIGEN is called only once.

DKNIGEN may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 1-26. DKNIGEN Return Codes

Code (dec)	Description
+00	Successful Completion.
+04	Item-Sequence Data Set Read Failure.
+08	Item-Sequence Data Set Write Failure.

DKNIGEN also issues the WTO message IGENxx, where xx is 01, 02 or 03, when it returns a non-zero completion code. These messages are documented in *CPCS-I Messages and Codes*.

Restrictions

DKNIGEN returns the sequence number eight last digits. The calling application is responsible for setting the other four digits (see "MICR Task Generation", in the *CPCS-I Programming Guide*, for information on the codes that are usually present in this area).

Example 1

Operation:

Get unique contiguous sequence numbers for a volume of 2000 codeline records. All sequence number first four digits are set to zeros.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSIGEN.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  IGEN-PARMS.
     05  IP-SORTER-NUMBER               PIC 9(01)  VALUE 0.
     05  IP-RECORD-VOLUME               PIC 9(04)  VALUE 2000.

 01  CURRENT-SEQUENCE-NUMBER.
     05  FILLER                         PIC 9(04)  VALUE 0.
     05  CS-SEQUENCE                    PIC 9(08).

 01  SUCCESSFUL-OPERATION               PIC 9(01)  VALUE 0.

 COPY DKNCRIGN.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                        PIC X(68).
/
 PROCEDURE DIVISION                     USING APTCB
                                              START-PARMS.

     ...................
     ...................
     ...................

     SET  IGN-FUNC-INSERT               TO TRUE.
     MOVE IP-SORTER-NUMBER              TO IGN-SORTER.
     MOVE IP-RECORD-VOLUME              TO IGN-VOL.

     CALL 'DKNIGEN'                     USING APTCB
                                              IGN.

     IF  RETURN-CODE                    =  SUCCESSFUL-OPERATION
         MOVE IGN-SEQNO                 TO CS-SEQUENCE
         PERFORM 1000-PROCESS-CODE-LINE
             IP-RECORD-VOLUME           TIMES.

     GOBACK.
```

Figure 1-37 (Part 1 of 2). DKNIGEN Example 1

```
  1000-PROCESS-CODE-LINE.

      ...................
      ...................
      ...................

      ADD 1                              TO CS-SEQUENCE.
```

Figure 1-37 (Part 2 of 2). DKNIGEN Example 1

Example 2

Operation:

Get unique sequence numbers for a variable number of codeline records. Sequence numbers do not need to be contiguous. All sequence number first four digits are set to zeros.

Assembler Code:

```
TSIGEN   CSECT
TSIGEN   AMODE 31
TSIGEN   RMODE ANY
*
         SAVE  (14,12)             SAVE REGISTERS
         BASR  R12,0               GET ROUTINE @
         USING *,R12               BASE SUBROUTINE
*
         LA    R14,SAVEAREA        POINT TO OUR SAVE AREA
         ST    R14,8(R13)          STORE FORWARD SAVE AREA @
         ST    R13,4(R14)          STORE BACKWARD SAVE AREA @
         LR    R13,R14             POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)        GET APTCB @
*
NEXTREC  MVC   IGPFUNCT,GETSEQ     SET 'GET SEQUENCE' REQUEST CODE
         LA    R1,PARMS            GET DKNIGEN PARM LIST @
         L     R15,IGEN@           GET DKNIGEN @
         BASSM R14,R15             CALL DKNIGEN TO GET SEQUENCE
         LTR   R15,R15             SUCCESSFUL OPERATION ?
         BNZ   IGENERR             N. PROCESS DKNIGEN ERROR
         .................
         .................
         .................
         B     RETURN              ALL RECORDS PROCESSED
*
         .................
         .................
         .................
         B     NEXTREC             PROCESS NEXT RECORD
*
```

Figure 1-38 (Part 1 of 2). DKNIGEN Example 2

```
IGENERR  DS   0H                    PROCESS DKNIGEN ERROR
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
GETSEQ   DC    CL2'I'               GET SEQUENCE NUMBER REQUEST CODE
*
IGEN@    DC    V(DKNIGEN)           DKNIGEN @
*
PARMS    DS   0F                    DKNIGEN PARM LIST
APTCB@   DS    A                    . APTCB @
         DC    A(IGENPARM)          . DKNIGEN CALL PARMS @
*
         DKNIGPRM TYPE=DATA         DKNIGEN CALL PARMS
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         END
```

Figure 1-38 (Part 2 of 2). DKNIGEN Example 2

Generating Spooled Reports

Before opening any output report data set, CPCS-I application tasks must call module DKNADCB2 to take advantage of the CPCS-I spool facilities, which make print functions device independent. See “Describing an Application Task’s Environment” in the *CPCS-I Customization Guide*, for a description of the DKNBLDL CLASS and PRINT parameters. See “DKNFORM - Printed Output Control”, in the *CPCS-I Terminal Operations Guide*, for information on CPCS-I spool and printer control.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to communicate with DKNADCB2:

Table 1-27. DKNADCB2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
FD (COBOL) / DCB (assembler) • An FD/DCB control block is required for each report data set to be processed by the same DKNADCB2 call.		
Report Description • This 32 byte description is shown, associated to the report name, on the FORM task ‘spools’ screen. • A description control block must be associated to each specified FD/DCB control block.		

DKNADCB2 may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 1-28. DKNADCB2 Return Codes

Code (dec)	Description
+00	Successful completion.
+08	No spool data sets available.

Restrictions

- COBOL programs must include ‘BLOCK CONTAINS 0’, ‘RECORDING MODE V’, ‘LABEL RECORDS STANDARD’ and an output record length of 132 bytes, in their report data set definition (FD) areas. Assembler programs must include DSORG=PS, RECFM=VB and LRECL=137 in their report data set definition (DCB) areas. Assembler programs must also place the record length (137), in binary, in bytes 1–2, and binary zeroes in bytes 3–4, preceding the 132 byte data area, on each output record, and use ASA format carriage control characters.

Important! Assembler programs calling DKNADCB2 must issue the FREEPOOL macro after closing a report data set. Failure to do so results in core fragmentation.

- If you successfully call DKNADCB2, you are then required to OPEN and CLOSE the spool file, even if you do not write to it.

Example 1

Operation:

Create a report using the CPCS-I spool facilities. 'REPORT 1' is the spool's report description.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSADCB2.
/
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

 FILE-CONTROL.

     SELECT REPORT1             ASSIGN DA-S-REP1.
/
 DATA DIVISION.
 FILE SECTION.

 FD  REPORT1
     BLOCK CONTAINS             0
     RECORDING MODE             V
     LABEL RECORDS              STANDARD.

 01  PRINT1-RECORD             PIC  X(132).
/
 WORKING-STORAGE SECTION.

 01  REPORT1-DESCRIPTION       PIC  X(032) VALUE 'REPORT 1'.

 01  LINE-COUNT                PIC  9(002) VALUE 0.
     88 FIRST-LINE                         VALUE 0.

 01  PAGE-LIMIT                PIC  9(002) VALUE 55.

 01  PRINT1-HEADING1.
     05  FILLER                PIC  X(043) VALUE ' '.
     05  FILLER                PIC  X(008) VALUE 'REPORT 1'.

 01  PRINT1-HEADING2.
     05  FILLER                PIC  X(004) VALUE ' '.
     05  FILLER                PIC  X(021) VALUE 'FIELD 1'.
     05  FILLER                PIC  X(019) VALUE 'FIELD 2'.
     05  FILLER                PIC  X(007) VALUE 'FIELD 3'.
```

Figure 1-39 (Part 1 of 3). DKNADCB2 Example 1

```
 01  PRINT1-DETAIL-LINE.
     05  P1D-FIELD1             PIC  X(015).
     05  FILLER                 PIC  X(010) VALUE ' '.
     05  P1D-FIELD2             PIC  X(007).
     05  FILLER                 PIC  X(010) VALUE ' '.
     05  P1D-FIELD3             PIC  X(011).

 01  BLANK-LINE                 PIC  X(132) VALUE ' '.

 01  REPORT1-COMPLETION-SW      PIC  X(001) VALUE 'N'.
     88  RC-REPORT1-COMPLETE                VALUE 'Y'.

 01  SUCCESSFUL-OPERATION       PIC  9(001) VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC  X(068).
/
 PROCEDURE DIVISION             USING APTCB
                                      START-PARMS.

     CALL 'DKNADCB2'                  USING APTCB
                                           REPORT1
                                           REPORT1-DESCRIPTION.

     IF  RETURN-CODE                  = SUCCESSFUL-OPERATION
         OPEN OUTPUT                  REPORT1

         PERFORM 1000-CREATE-REPORT1
             UNTIL RC-REPORT1-COMPLETE

         CLOSE                        REPORT1.

     GOBACK.

 1000-CREATE-REPORT1.

     IF  FIRST-LINE
      OR LINE-COUNT                   >  PAGE-LIMIT
         WRITE PRINT1-RECORD          FROM PRINT1-HEADING1
             AFTER ADVANCING          PAGE
         WRITE PRINT1-RECORD          FROM PRINT1-HEADING2
             AFTER ADVANCING          2
         WRITE PRINT1-RECORD          FROM BLANK-LINE
             AFTER ADVANCING          1
         MOVE 4                       TO LINE-COUNT.

     IF  .................

         SET RC-REPORT1-COMPLETE      TO TRUE

     ELSE
```

Figure 1-39 (Part 2 of 3). DKNADCB2 Example 1

```
         .................
         .................
         .................

         MOVE .................      TO P1D-FIELD1
         MOVE .................      TO P1D-FIELD2
         MOVE .................      TO P1D-FIELD3
         WRITE PRINT1-RECORD         FROM PRINT1-DETAIL-LINE
             AFTER ADVANCING         1
         ADD 1                       TO LINE-COUNT.
```

Figure 1-39 (Part 3 of 3). DKNADCB2 Example 1

Example 2

Operation:

Create two reports using the CPCS-I spool facilities. 'REPORT 1' and 'REPORT 2' are the respective spool's report descriptions.

Assembler Code:

```
TSADCB2  CSECT
TSADCB2  AMODE 24
TSADCB2  RMODE 24
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)          GET APTCB @
         OI    PARMSEND,ENDLIST      SET 'END OF LIST' FLAG
         LA    R1,PARMS              GET DKNADCB2 PARM LIST @
         L     R15,ADCB2@            GET DKNADCB2 @
         BASSM R14,R15               CALL DKNADCB2
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BNZ   ADCB2ERR              N. PROCESS ADCB2 ERROR
*
         OPEN  (DCB1,(OUTPUT))       OPEN REPORT 1 DCB
         OPEN  (DCB2,(OUTPUT))       OPEN REPORT 2 DCB
*
NEXTREC  DS   0H                     PROCESS NEXT RECORD
         .............
         .............
         .............
         B     BUILDRP1              PRINT REPORT 1 RECORD
*
```

Figure 1-40 (Part 1 of 4). DKNADCB2 Example 2

```
         .................
         .................
         .................
         B     BUILDRP2                PRINT REPORT 2 RECORD
*
         .................
         .................
         .................
         B     REPEND                  ALL RECORDS PROCESSED
*
BUILDRP1 MVC   REP1FLD1,.............   SET REPORT 1 FIELD 1
         MVC   REP1FLD2,.............   SET REPORT 1 FIELD 2
         CP    LINECNT1,LCFIRST        REPORT 1 FIRST LINE ?
         BE    WRITEHD1                Y. WRITE HEADINGS
         CP    LINECNT1,LCPGMAX        END OF PAGE REACHED ?
         BL    WRITELN1                Y. WRITE LINE
*
WRITEHD1 MVC   OUDATA,REP1HDG1         GET REPORT 1 HEADING 1
         PUT   DCB1,OUTREC             WRITE REPORT 1 HEADING 1
         MVC   OUDATA,REP1HDG2         GET REPORT 1 HEADING 2
         PUT   DCB1,OUTREC             WRITE REPORT 1 HEADING 2
         MVC   OUDATA,BLNKLINE         GET BLANK LINE
         PUT   DCB1,OUTREC             SKIP 1 LINE
         ZAP   LINECNT1,LCHDGINC       INITIALIZE REPORT 1 LINE CTR
*
WRITELN1 MVC   OUDATA,REP1DET          GET REPORT 1 DETAIL LINE
         PUT   DCB1,OUTREC             WRITE REPORT 1 DETAIL LINE
         AP    LINECNT1,LCDETINC       INCREMENT REPORT 1 LINE CTR
         B     NEXTREC                 PROCESS NEXT RECORD
*
BUILDRP2 MVC   REP2FLD1,.............   SET REPORT 2 FIELD 1
         CP    LINECNT2,LCFIRST        REPORT 2 FIRST LINE ?
         BE    WRITEHD2                Y. WRITE HEADINGS
         CP    LINECNT2,LCPGMAX        END OF PAGE REACHED ?
         BL    WRITELN2                N. WRITE DETAIL LINE
*
WRITEHD2 MVC   OUDATA,REP2HDG1         GET REPORT 2 HEADING 1
         PUT   DCB2,OUTREC             WRITE REPORT 2 HEADING 1
         MVC   OUDATA,BLNKLINE         GET BLANK LINE
         PUT   DCB2,OUTREC             SKIP 1 LINE
         ZAP   LINECNT2,LCHDGINC       INITIALIZE REPORT 2 LINE CTR
*
WRITELN2 MVC   OUDATA,REP2DET          GET REPORT 2 DETAIL LINE
         PUT   DCB2,OUTREC             WRITE REPORT 2 DETAIL LINE
         AP    LINECNT2,LCDETINC       INCREMENT REPORT 2 LINE CTR
         B     NEXTREC                 PROCESS NEXT RECORD
*
ADCB2ERR DS    0H                      PROCESS DKNADCB2 ERROR
         .................
         .................
         .................
         B     RETURN                  EXIT PROGRAM
*
```

Figure 1-40 (Part 2 of 4). DKNADCB2 Example 2

```
REPEND   CLOSE (DCB1)                 CLOSE REPORT 1 DCB
         CLOSE (DCB2)                 CLOSE REPORT 2 DCB
         FREEPOOL DCB1                FREE DCB1 BUFFER POOL
         FREEPOOL DCB2                FREE DCB2 BUFFER POOL
*
RETURN   L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
REP1HDG1 DS   0H                      REPORT 1 HEADING 1
         DC    C'1'                   . NEW PAGE CARRIAGE CNTRL
         DC  43C' '                   . FILLER
         DC    C'REPORT 1'            . TITLE
         DC  81C' '                   . FILLER
*
REP1HDG2 DS   0H                      REPORT 1 HEADING 2
         DC    C'0'                   . DOUBLE SPACE CARRIAGE CNTRL
         DC   7C' '                   . FILLER
         DC    C'FIELD 1'             . TITLE
         DC   7C' '                   . FILLER
         DC    C'FIELD 2'             . TITLE
         DC 104C' '                   . FILLER
*
REP2HDG1 DS   0H                      REPORT 2 HEADING 1
         DC    C'1'                   . NEW PAGE CARRIAGE CNTRL
         DC  43C' '                   . FILLER
         DC    C'REPORT 2'            . TITLE
         DC  81C' '                   . FILLER
*
BLNKLINE DS   0H                      BLANK LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC 132C' '                   . FILLER
*
REP1DET  DS   0H                      REPORT 1 DETAIL LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC   5C' '                   . FILLER
REP1FLD1 DS    CL11                   . FIELD 1
         DC   5C' '                   . FILLER
REP1FLD2 DS    CL7                    . FIELD 2
         DC 104C' '                   . FILLER
*
REP2DET  DS   0H                      REPORT 2 DETAIL LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC   5C' '                   . FILLER
REP2FLD1 DS    CL11                   . FIELD 1
         DC 116C' '                   . FILLER
*
         DS   0H
OUTREC   DS   0XL137                  OUTPUT RECORD
         DC    H'137'                 . RECORD LENGTH
         DC    H'0'                   . FILLER
OUDATA   DS    CL133                  . DATA
*
ADCB2@   DC    V(DKNADCB2)            DKNADCB2 @
*
```

Figure 1-40 (Part 3 of 4). DKNADCB2 Example 2

```
ENDLIST  EQU   X'80'                   'END OF LIST' FLAG
*
LINECNT1 DC    P'0'                    REPORT 1 LINE COUNTER
LINECNT2 DC    P'0'                    REPORT 2 LINE COUNTER
LCFIRST  DC    P'0'                    . INITIAL VALUE
LCDETINC DC    P'1'                    . DETAIL LINE INCREMENT VALUE
LCHDGINC DC    P'4'                    . HEADINGS INCREMENT VALUE
LCPGMAX  DC    P'55'                   . LINES IN PAGE
*
PARMS    DS   0F                       DKNADCB2 PARM LIST
APTCB@   DS    A                       . APTCB @
         DC    A(DCB1)                 . DCB1 @
         DC    A(DCB1DESC)             . DCB1 DESCRIPTION @
         DC    A(DCB2)                 . DCB2 @
PARMSEND DC    A(DCB2DESC)             . DCB2 DESCRIPTION @
*
DCB1DESC DC    CL32'REPORT 1'          REPORT 1 DESCRIPTION
DCB2DESC DC    CL32'REPORT 2'          REPORT 2 DESCRIPTION
*
DCB1     DCB   DSORG=PS,DDNAME=REP1,RECFM=VB,LRECL=137,MACRF=PM
DCB2     DCB   DSORG=PS,DDNAME=REP2,RECFM=VB,LRECL=137,MACRF=PM
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
         END
```

Figure 1-40 (Part 4 of 4). DKNADCB2 Example 2

Generating JES Reports

Before opening the report output data sets, CPCS-I tasks may call module DKNADCB3 to have their print output sent directly to JES. If so, JES dynamically allocates SYSOUT data sets for the reports.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to communicate with DKNADCB3:

Table 1-29. DKNADCB3 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB

FD Name / DCB (assembler)

- An FD/DCB control block is required for each report data set to be processed by the same DKNADCB3 call.

DKNADCB3 may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 1-30. DKNADCB3 Return Codes

Code (dec)	Description
+00	Successful completion.
+04	Maximum number of FDs/DCBs exceeded.
+08	SYSOUT data set dynamic allocation error.

Restrictions

The DKNBLDL PRINT parameter must not be specified for tasks calling DKNADCB3.

COBOL programs must include ‘BLOCK CONTAINS 0’, ‘RECORDING MODE V’, ‘LABEL RECORDS STANDARD’ and an output record length of 132 bytes, in their report data set definition (FD) areas. Assembler programs must include DSORG=PS, RECFM=VB and LRECL=137 in their report data set definition (DCB) areas. Assembler programs must also place the record length (137), in binary, in bytes 1–2, and binary zeroes in bytes 3–4, preceding the 132 byte data area, on each output record, and use ASA carriage control characters.

Example 1

Operation:

Create a report that is to be written to JES.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSADCB3.
/
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

 FILE-CONTROL.

     SELECT REPORT1             ASSIGN DA-S-REP1.
/
 DATA DIVISION.
 FILE SECTION.

 FD  REPORT1
     BLOCK CONTAINS             0
     RECORDING MODE             V
     LABEL RECORDS              STANDARD.

 01  PRINT1-RECORD             PIC  X(132).
/
 WORKING-STORAGE SECTION.

 01  LINE-COUNT                PIC  9(002) VALUE 0.
     88 FIRST-LINE                        VALUE 0.

 01  PAGE-LIMIT                PIC  9(002) VALUE 55.

 01  PRINT1-HEADING1.
     05  FILLER                PIC  X(043) VALUE ' '.
     05  FILLER                PIC  X(008) VALUE 'REPORT 1'.

 01  PRINT1-HEADING2.
     05  FILLER                PIC  X(004) VALUE ' '.
     05  FILLER                PIC  X(021) VALUE 'FIELD 1'.
     05  FILLER                PIC  X(019) VALUE 'FIELD 2'.
     05  FILLER                PIC  X(007) VALUE 'FIELD 3'.

 01  PRINT1-DETAIL-LINE.
     05  P1D-FIELD1            PIC  X(015).
     05  FILLER                PIC  X(010) VALUE ' '.
     05  P1D-FIELD2            PIC  X(007).
     05  FILLER                PIC  X(010) VALUE ' '.
     05  P1D-FIELD3            PIC  X(011).

 01  BLANK-LINE                PIC  X(132) VALUE ' '.
```

Figure 1-41 (Part 1 of 2). DKNADCB3 Example 1

```
 01  REPORT1-COMPLETION-SW      PIC  X(001) VALUE 'N'.
     88  RC-REPORT1-COMPLETE               VALUE 'Y'.

 01  SUCCESSFUL-OPERATION       PIC  9(001) VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC  X(068).
/
 PROCEDURE DIVISION             USING APTCB
                                      START-PARMS.

     CALL 'DKNADCB3'                  USING APTCB
                                           REPORT1.

     IF  RETURN-CODE                  = SUCCESSFUL-OPERATION
         OPEN OUTPUT                  REPORT1

         PERFORM 1000-CREATE-REPORT1
             UNTIL RC-REPORT1-COMPLETE

         CLOSE                        REPORT1.

     GOBACK.

 1000-CREATE-REPORT1.

     IF  FIRST-LINE
      OR LINE-COUNT                   >  PAGE-LIMIT
         WRITE PRINT1-RECORD          FROM PRINT1-HEADING1
             AFTER ADVANCING          PAGE
         WRITE PRINT1-RECORD          FROM PRINT1-HEADING2
             AFTER ADVANCING          2
         WRITE PRINT1-RECORD          FROM BLANK-LINE
             AFTER ADVANCING          1
         MOVE 4                       TO LINE-COUNT.

     IF  .................

         SET RC-REPORT1-COMPLETE      TO TRUE

     ELSE

         .................
         .................
         .................

         MOVE .................       TO P1D-FIELD1
         MOVE .................       TO P1D-FIELD2
         MOVE .................       TO P1D-FIELD3
         WRITE PRINT1-RECORD          FROM PRINT1-DETAIL-LINE
             AFTER ADVANCING          1
         ADD 1                        TO LINE-COUNT.
```

Figure 1-41 (Part 2 of 2). DKNADCB3 Example 1

Example 2

Operation:

Create two reports that are to be written to JES. 'REPORT 1' and 'REPORT 2' are the respective spool report descriptions.

Assembler Code:

```
TSADCB3  CSECT
TSADCB3  AMODE 24
TSADCB3  RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)           GET APTCB @
         OI    PARMSEND,ENDLIST       SET 'END OF LIST' FLAG
         LA    R1,PARMS               GET DKNADCB3 PARM LIST @
         L     R15,ADCB3@             GET DKNADCB3 @
         BASSM R14,R15                CALL DKNADCB3
         LTR   R15,R15                SUCCESSFUL OPERATION ?
         BNZ   ADCB3ERR               N. PROCESS ADCB3 ERROR
*
         OPEN  (DCB1,(OUTPUT))        OPEN REPORT 1 DCB
         OPEN  (DCB2,(OUTPUT))        OPEN REPORT 2 DCB
*
NEXTREC  DS   0H                      PROCESS NEXT RECORD
         .................
         .................
         .................
         B     BUILDRP1               PRINT REPORT 1 RECORD
*
         .................
         .................
         .................
         B     BUILDRP2               PRINT REPORT 2 RECORD
*
         .................
         .................
         .................
         B     REPEND                 ALL RECORDS PROCESSED
*
```

Figure 1-42 (Part 1 of 4). DKNADCB3 Example 2

```
BUILDRP1 MVC   REP1FLD1,.....          SET REPORT 1 FIELD 1
         MVC   REP1FLD2,.....          SET REPORT 1 FIELD 2
         CP    LINECNT1,LCFIRST        REPORT 1 FIRST LINE ?
         BE    WRITEHD1                Y. WRITE HEADINGS
         CP    LINECNT1,LCPGMAX        END OF PAGE REACHED ?
         BL    WRITELN1                N. WRITE DETAIL LINE
*
WRITEHD1 MVC   OUDATA,REP1HDG1         GET REPORT 1 HEADING 1
         PUT   DCB1,OUTREC             WRITE REPORT 1 HEADING 1
         MVC   OUDATA,REP1HDG2         GET REPORT 1 HEADING 2
         PUT   DCB1,OUTREC             WRITE REPORT 1 HEADING 2
         MVC   OUDATA,BLNKLINE         GET BLANK LINE
         PUT   DCB1,OUTREC             SKIP 1 LINE
         ZAP   LINECNT1,LCHDGINC       INITIALIZE REPORT 1 LINE CTR
*
WRITELN1 MVC   OUDATA,REP1DET          GET REPORT 1 DETAIL LINE
         PUT   DCB1,OUTREC             WRITE REPORT 1 DETAIL LINE
         AP    LINECNT1,LCDETINC       INCREMENT REPORT 1 LINE CTR
         B     NEXTREC                 PROCESS NEXT RECORD
*
BUILDRP2 MVC   REP2FLD1,.....          SET REPORT 2 FIELD 1
         CP    LINECNT2,LCFIRST        REPORT 2 FIRST LINE ?
         BE    WRITEHD2                Y. WRITE HEADINGS
         CP    LINECNT2,LCPGMAX        END OF PAGE REACHED ?
         BL    WRITELN2                Y. WRITE LINE
*
WRITEHD2 MVC   OUDATA,REP2HDG1         GET REPORT 2 HEADING 1
         PUT   DCB2,OUTREC             WRITE REPORT 2 HEADING 1
         MVC   OUDATA,BLNKLINE         GET BLANK LINE
         PUT   DCB2,OUTREC             SKIP 1 LINE
         ZAP   LINECNT2,LCHDGINC       INITIALIZE REPORT 2 LINE CNTR
*
WRITELN2 MVC   OUDATA,REP2DET          GET REPORT 2 DETAIL LINE
         PUT   DCB2,OUTREC             WRITE REPORT 2 DETAIL LINE
         AP    LINECNT2,LCDETINC       INCREMENT REPORT 2 LINE CTR
         B     NEXTREC                 PROCESS NEXT RECORD
*
ADCB3ERR DS   0H                       PROCESS DKNADCB3 ERROR
         .............
         .............
         .............
         B     RETURN                  EXIT PROGRAM
*
REPEND   CLOSE (DCB1)                  CLOSE REPORT 1 DCB
         CLOSE (DCB2)                  CLOSE REPORT 2 DCB
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
```

Figure 1-42 (Part 2 of 4). DKNADCB3 Example 2

```
REP1HDG1 DS   0H                      REPORT 1 HEADING 1
         DC    C'1'                   . NEW PAGE CARRIAGE CNTRL
         DC  43C' '                   . FILLER
         DC    C'REPORT 1'            . TITLE
         DC  81C' '                   . FILLER
*
REP1HDG2 DS   0H                      REPORT 1 HEADING 2
         DC    C'0'                   . DOUBLE SPACE CARRIAGE CNTRL
         DC   7C' '                   . FILLER
         DC    C'FIELD 1'             . TITLE
         DC   7C' '                   . FILLER
         DC    C'FIELD 2'             . TITLE
         DC 104C' '                   . FILLER
*
REP2HDG1 DS   0H                      REPORT 2 HEADING 1
         DC    C'1'                   . NEW PAGE CARRIAGE CNTRL
         DC  43C' '                   . FILLER
         DC    C'REPORT 2'            . TITLE
         DC  81C' '                   . FILLER
*
BLNKLINE DS   0H                      BLANK LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC 132C' '                   . FILLER
*
REP1DET  DS   0H                      REPORT 1 DETAIL LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC   5C' '                   . FILLER
REP1FLD1 DS    CL11                   . FIELD 1
         DC   5C' '                   . FILLER
REP1FLD2 DS    CL7                    . FIELD 2
         DC 104C' '                   . FILLER
*
REP2DET  DS   0H                      REPORT 2 DETAIL LINE
         DC    C' '                   . SINGLE SPACE CARRIAGE CNTRL
         DC   5C' '                   . FILLER
REP2FLD1 DS    CL11                   . FIELD 1
         DC 116C' '                   . FILLER
*
         DS   0H
OUTREC   DS   0XL137                  OUTPUT RECORD
         DC    H'137'                 . RECORD LENGTH
         DC    H'0'                   . FILLER
OUDATA   DS    CL133                  . DATA
*
ADCB3@   DC    V(DKNADCB3)            DKNADCB3 @
*
ENDLIST  EQU   X'80'                  'END OF LIST' FLAG
*
LINECNT1 DC    P'0'                   REPORT 1 LINE COUNTER
LINECNT2 DC    P'0'                   REPORT 2 LINE COUNTER
LCFIRST  DC    P'0'                   . INITIAL VALUE
LCDETINC DC    P'1'                   . DETAIL LINE INCREMENT VALUE
LCHDGINC DC    P'4'                   . HEADINGS INCREMENT VALUE
LCPGMAX  DC    P'55'                  . LINES IN PAGE
*
```

Figure 1-42 (Part 3 of 4). DKNADCB3 Example 2

```
PARMS    DS   0F                       DKNADCB3 PARM LIST
APTCB@   DS    A                       . APTCB @
         DC    A(DCB1)                 . DCB1 @
PARMSEND DC    A(DCB2)                 . DCB2 @
*
DCB1     DCB   DSORG=PS,DDNAME=REP1,RECFM=VB,LRECL=137,MACRF=PM
DCB2     DCB   DSORG=PS,DDNAME=REP2,RECFM=VB,LRECL=137,MACRF=PM
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
         END
```

Figure 1-42 (Part 4 of 4). DKNADCB3 Example 2

Generating Columnar Reports

CPCS-I performs automatic full-page buffering for applications generating columnar reports. This function allows reports to process sequentially without the application program buffering that is necessary to generate this type of report.

Programs request the full-page buffering function by writing a mask record that describes how buffering must be performed. The mask record must be the first record written and it must be written only once. Mask records have the following format:

Table 1-31 (Page 1 of 2). Full-Page Buffering Mask Record Format

Bytes	Description
01	Full-Page Buffering Request Code • This field must contain the value X'00'.
02–05	Special Character Translate Table • This field is only used for X'00' detail record formats (see detail record formats below). It contains the characters that characters B, C, D and E must be respectively translated to. The character A is always translated to a blank.
06	Number of Columns • This binary field indicates how many columns are included on each page (for example two-up printing, or three-up printing).
07	Number of Detail Lines • This binary field indicates how many detail lines are printed on each page for a single column.
08	Print Record Blocking Factor • This binary field indicates the number of data records contained in each logical record passed by the application program.
09	Record Length • This binary field indicates the length of the detail records passed by the application program. It does not include the length of the control fields (first two bytes) of X'00' type detail records.

Table 1-31 (Page 2 of 2). Full-Page Buffering Mask Record Format

Bytes	Description
10–41	Print Edit Information • This variable length field specifies the edit mask information used to format each column-item detail data that follows this mask record. This field is only used for X'00' type detail records. The last byte must be X'FF'. The mask may have a maximum length of 32 bytes, and is processed sequentially from left to right. The format of each byte is: **Bits 0–3** **Function identifier** **0** Ignore input digits for the length in bits 4–7. **1** Use input digits for the length in bits 4–7, suppressing leading zeroes. **3** Use input digits for the length in bits 4–7, printing leading zeroes. **A** Insert a blank or blanks at the current buffer location for the length in bits 4–7. **B, C, D, E** Insert the respective user-assigned characters, as described in the special character translate table, for the length in bits 4–7. **Bits 4–7** **Print positions** This is the number of print positions minus 1 in binary.

Once the mask record has been written, the application task can write three types of records:

Header/Trailer Records

These are full 132-character lines of alphanumeric information printed before and after the columnar information. These records are identified by any of the following carriage control codes in their first byte:

Table 1-32. Full-Page Buffering Header/Trailer Carriage Control Codes

Code (hex)	Description
+09	Space one line after printing.
+11	Space two lines after printing.
+19	Space three lines after printing.
+89	Skip to top of page after printing.

Compressed Format Detail Records

These types of detail records are identified by a value of X'00' in their first byte. If the record is full, that is, the number of column-item data elements equals the specification in the eighth byte of the mask record, the second byte must also be X'00'. If there are fewer than this number of bytes, the second byte must contain the actual binary number of bytes in the record.

The detail data must be in packed unsigned format. Valid characters are 0 through 9 and A, B, C, D, E. These characters are printed/translated as specified in the mask record character translate table and print edit information.

Character Format Detail Records

These types of detail records are identified by a value of X'F1' or X'F2' in their first byte. The detail data must be in character format. These records are placed directly in the column without any editing, allowing the imbedding of alphabetic data in the column-item data. The length of the X'F1' record is the record length specified in the ninth byte of the mask record. The length of the X'F2' record is twice the record length specified in the ninth byte of the mask record.

Restrictions

Full-page buffering is available only to programs using the CPCS-I spooled/JES report functions (see “Generating Spooled Reports” on page 1-90 and “Generating JES Reports” on page 1-97).

Example 1

Operation:

Create the columnar report described by the following layout:

```
                          REPORT 1

 FIELD 1    FIELD 2                        FIELD 1    FIELD 2

XXX-XXXXX ZZZZZZZ.99-                    XXX-XXXXX ZZZZZZZ.99-
XXX-XXXXX ZZZZZZZ.99-                    XXX-XXXXX ZZZZZZZ.99-

......... ...........                    ......... ...........
......... ...........
......... ...........                    XXX-XXXXX ZZZZZZZ.99-

XXX-XXXXX ZZZZZZZ.99-
```

Characters B, C, D and E are respectively translated to . , - and *. Each page contains a maximum of 54 detail lines.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSCOLM.
/
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

 FILE-CONTROL.
```

Figure 1-43 (Part 1 of 4). Columnar Report Example 1

```
     SELECT REPORT1             ASSIGN DA-S-REP1.
/
 DATA DIVISION.
 FILE SECTION.

 FD  REPORT1
     BLOCK CONTAINS             0
     RECORDING MODE             V
     LABEL RECORDS              STANDARD.

 01  PRINT1-RECORD             PIC  X(132).
/
 WORKING-STORAGE SECTION.

 01  REPORT1-DESCRIPTION       PIC  X(032) VALUE 'REPORT 1'.

 01  LINE-COUNT                PIC  9(002) VALUE 0.
     88 LC-FIRST-LINE                      VALUE 0.

 01  PAGE-LIMIT                PIC  9(003) VALUE 108.

 01  EDIT-MASK-RECORD.
     05  EM-FULL-PAGE-BUFF-REQU PIC  X(001) VALUE X'00'.

     05  EM-B-TRANSLATE-CHAR   PIC  X(001) VALUE '.'.
     05  EM-C-TRANSLATE-CHAR   PIC  X(001) VALUE ','.
     05  EM-D-TRANSLATE-CHAR   PIC  X(001) VALUE '-'.
     05  EM-E-TRANSLATE-CHAR   PIC  X(001) VALUE '*'.

     05  EM-COLUMNS-PER-PAGE   PIC  X(001) VALUE X'02'.
     05  EM-LINES-PER-PAGE     PIC  X(001) VALUE X'36'.
     05  EM-COLUMNS-PER-RECORD PIC  X(001) VALUE X'01'.
     05  EM-RECORD-LENGTH      PIC  X(001) VALUE X'09'.

     05  EM-FLD1-LNGDZERO-3CHAR PIC  X(001) VALUE X'32'.
     05  EM-FLD1-HIPHEN-1CHAR  PIC  X(001) VALUE X'D0'.
     05  EM-FLD1-LNGDZERO-4CHAR PIC  X(001) VALUE X'34'.
     05  EM-BLANK-1CHAR        PIC  X(001) VALUE X'A0'.
     05  EM-FLD2-SPRSZERO-7CHAR PIC  X(001) VALUE X'16'.
     05  EM-FLD2-PERIOD-1CHAR  PIC  X(001) VALUE X'B0'.
     05  EM-FLD2-LDNGZERO-3CHAR PIC  X(001) VALUE X'32'.
     05  EM-EDIT-MASK-END      PIC  X(001) VALUE X'FF'.

 01  PRINT1-HEADING1.
     05  PH1-CARRIAGE-CONTROL  PIC  X(001) VALUE X'11'.
     05  FILLER                PIC  X(034) VALUE ' '.
     05  FILLER                PIC  X(008) VALUE 'REPORT 1'.

 01  PRINT1-HEADING2.
     05  PH2-CARRIAGE-CONTROL  PIC  X(001) VALUE X'11'.
     05  FILLER                PIC  X(001) VALUE ' '.
     05  FILLER                PIC  X(011) VALUE 'FIELD 1'.
     05  FILLER                PIC  X(055) VALUE 'FIELD 2'.
     05  FILLER                PIC  X(011) VALUE 'FIELD 1'.
```

Figure 1-43 (Part 2 of 4). Columnar Report Example 1

```
     05  FILLER                PIC  X(007) VALUE 'FIELD 2'.

 01  PRINT1-DETAIL-COLUMN.
     05  P1D-DETAIL-RECORD-ID   PIC  X(001) VALUE X'00'.
     05  P1D-ELEMENT-INDICATOR  PIC  X(001) VALUE X'00'.
     05  P1D-FIELD1-PACKED-UNS  PIC  X(004).
     05  P1D-FIELD2-PACKED-UNS  PIC  X(005).

 01  BLANK-LINE.
     05  BL-CARRIAGE-CONTROL    PIC  X(001) VALUE X'89'.
     05  FILLER                 PIC  X(132) VALUE ' '.

 01  REPORT1-COMPLETION-SW      PIC  X(001) VALUE 'N'.
     88  RC-REPORT1-COMPLETE                VALUE 'Y'.

 01  SUCCESSFUL-OPERATION       PIC  9(001) VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC  X(068).
/
 PROCEDURE DIVISION             USING APTCB
                                      START-PARMS.

     CALL 'DKNADCB2'                  USING APTCB
                                            REPORT1
                                            REPORT1-DESCRIPTION.

     IF  RETURN-CODE                  = SUCCESSFUL-OPERATION
         OPEN OUTPUT                  REPORT1

         WRITE PRINT1-RECORD          FROM EDIT-MASK-RECORD
         PERFORM 1000-CREATE-REPORT1
             UNTIL RC-REPORT1-COMPLETE

         CLOSE                        REPORT1.

     GOBACK.

 1000-CREATE-REPORT1.

     IF  ......................
         SET RC-REPORT1-COMPLETE      TO TRUE
     ELSE
         IF  LC-FIRST-LINE
          OR LINE-COUNT               >  PAGE-LIMIT
             WRITE PRINT1-RECORD      FROM BLANK-LINE
             WRITE PRINT1-RECORD      FROM PRINT1-HEADING1
             WRITE PRINT1-RECORD      FROM PRINT1-HEADING2
             SET LC-FIRST-LINE        TO TRUE
         END-IF
```

Figure 1-43 (Part 3 of 4). Columnar Report Example 1

```
         MOVE ..............          TO P1D-FIELD1-PACKED-UNS
         MOVE ..............          TO P1D-FIELD2-PACKED-UNS
         WRITE PRINT1-RECORD          FROM PRINT1-DETAIL-COLUMN

         ADD 1                        TO LINE-COUNT.
```

Figure 1-43 (Part 4 of 4). Columnar Report Example 1

Example 2

Operation:

Create the columnar report described by the following layout:

```
                          REPORT 1

 FIELD 1    FIELD 2   TYPE           FIELD 1    FIELD 2   TYPE

 XXX-XXXXX 9999999.99 MISSING        XXX-XXXXX 9999999.99 FREE
 XXX-XXXXX 9999999.99 FREE           XXX-XXXXX 9999999.99 FREE

 ......... ..........   .           ......... ..........   .
 ......... ..........   .
 ......... ..........   .           XXX-XXXXX 9999999.99 MISSING

 XXX-XXXXX 9999999.99 MISSING
```

Each page contains a maximum of 40 detail lines.

Assembler Code:

```
TSCOLM   CSECT
TSCOLM   AMODE 24
TSCOLM   RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
```

Figure 1-44 (Part 1 of 4). Columnar Report Example 2

```
         MVC   APTCB@,0(R1)            GET APTCB @
         OI    PARMSEND,ENDLIST        SET 'END OF LIST' FLAG
         LA    R1,PARMS                GET DKNADCB2 PARM LIST @
         L     R15,ADCB2@              GET DKNADCB2 @
         BASSM R14,R15                 CALL DKNADCB2
         LTR   R15,R15                 SUCCESSFUL OPERATION ?
         BNZ   ADCB2ERR                N. PROCESS ADCB2 ERROR
*
         OPEN  (DCB1,(OUTPUT))         OPEN REPORT 1 DCB
*
         MVC   OUDATA,EDMASK           GET EDIT MASK RECORD
         PUT   DCB1,OUTREC             PASS EDIT MASK RECORD
*
NEXTREC  DS   0H                       PROCESS NEXT RECORD
         .................
         .................
         .................
         B     BUILDRP1                PRINT REPORT RECORD
*
         .................
         .................
         .................
         B     REPEND                  ALL RECORDS PROCESSED
*
BUILDRP1 MVC   REP1FLD1,.............
         MVC   REP1FLD2,.............
         MVC   REP1FLD3,=C'MISSING'
         CP    LINECNT,LCFIRST         REPORT FIRST LINE ?
         BE    WRITEHDS                Y. WRITE HEADINGS
         CP    LINECNT,LCPGMAX         END OF PAGE REACHED ?
         BL    WRITELNE                N. DETAIL WRITE LINE
*
WRITEHDS MVC   OUDATA,BLNKLINE         GET BLANK LINE
         PUT   DCB1,OUTREC             START PAGE
         MVC   OUDATA,REP1HDG1         GET REPORT HEADING 1
         PUT   DCB1,OUTREC             WRITE REPORT HEADING 1
         MVC   OUDATA,REP1HDG2         GET REPORT HEADING 2
         PUT   DCB1,OUTREC             WRITE REPORT HEADING 2
         ZAP   LINECNT,LCFIRST         INITIALIZE LINE COUNTER
*
WRITELNE MVC   OUDATA,REP1DET          GET DETAIL LINE
         PUT   DCB1,OUTREC             WRITE DETAIL LINE
         AP    LINECNT,LCDETINC        INCREMENT LINE COUNTER
         B     NEXTREC                 PROCESS NEXT RECORD
*
ADCB2ERR DS   0H                       PROCESS DKNADCB2 ERROR
         .................
         .................
         .................
         B     RETURN                  EXIT PROGRAM
*
REPEND   CLOSE (DCB1)                  CLOSE REPORT 1 DCB
         FREEPOOL DCB1                 FREE DCB1 BUFFER POOL
*
```

Figure 1-44 (Part 2 of 4). Columnar Report Example 2

```
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN
*
         DKNREGS                       REGISTER EQUATES
*
EDMASK   DS   0H                       EDIT MASK RECORD
         DC    X'00'                   . FULL PAGE BUFFERING REQUEST
         DC   4C' '                    . NOT USED
         DC    X'03'                   . COLUMNS PER PAGE
         DC    X'28'                   . LINES PER PAGE
         DC    X'01'                   . COLUMNS PER RECORD
         DC    X'1D'                   . RECORD LENGTH
*
REP1HDG1 DS   0H                       REPORT HEADING 1
         DC    X'11'                   . CARRIAGE CONTROL
         DC  34C' '                    . FILLER
         DC    C'REPORT 1'             . TITLE
         DC  81C' '                    . FILLER
*
REP1HDG2 DS   0H                       REPORT HEADING 2
         DC    X'11'                   . CARRIAGE CONTROL
         DC    CL2' '                  . FILLER
         DC    CL11'FIELD 1'           . FIELD 1 TITLE
         DC    CL10'FIELD 2'           . FIELD 2 TITLE
         DC    CL110'TYPE'             . FIELD 3 TITLE
*
BLNKLINE DS   0H                       BLANK LINE
         DC    X'89'                   . CARRIAGE CONTROL
         DC 132C' '                    . FILLER
*
REP1DET  DS   0H                       REPORT 1 DETAIL LINE
         DC    X'F1'                   . DETAIL RECORD TYPE - FORMATD
         DC    C' '                    . FILLER
REP1FLD1 DS    CL9                     . FIELD 1
         DC    C' '                    . FILLER
REP1FLD2 DS    CL10                    . FIELD 2
         DC    C' '                    . FILLER
REP1FLD3 DS    CL7                     . FIELD 3
         DC 103C' '                    . FILLER
*
         DS   0H
OUTREC   DS   0XL137                   OUTPUT RECORD
         DC    H'137'                  . RECORD LENGTH
         DC    H'0'                    . FILLER
OUDATA   DS    CL133                   . DATA
*
ADCB2@   DC    V(DKNADCB2)             DKNADCB2 @
*
ENDLIST  EQU   X'80'                   'END OF LIST' FLAG
*
LINECNT  DC    P'0'                    LINE COUNTER
LCFIRST  DC    P'0'                    . INITIAL VALUE
LCDETINC DC    P'1'                    . DETAIL LINE INCREMENT VALUE
LCPGMAX  DC    P'40'                   . LINES IN PAGE
*
```

Figure 1-44 (Part 3 of 4). Columnar Report Example 2

```
PARMS    DS   0F                       DKNAPTCB2 PARM LIST
APTCB@   DS    A                       . APTCB @
         DC    A(DCB1)                 . DCB1 @
PARMSEND DC    A(DCB1DESC)             . DCB1 DESCRIPTION @
*
DCB1DESC DC    CL32'REPORT 1'          REPORT 1 DESCRIPTION
*
DCB1     DCB   DSORG=PS,DDNAME=REP1,RECFM=VB,LRECL=137,MACRF=PM
*
SAVEAREA DS  18F                       OUR SAVE AREA
*
         END
```

Figure 1-44 (Part 4 of 4). Columnar Report Example 2

Generating Expanded Codeline Record Reports

Expanded (customized) codeline records may be printed by calling module DKNMDXR, a standard print interface that automatically aligns captions and prints multiple detail lines per record as required.

DKNMDXR may also be used to print standard length codeline records. Data sets are not defined in the application program (no FDs/DCBs are required). Page break processing is also automatic. Reports are generated as CPCS-I spooled reports, but DKNADCB2 does not need to be called by the application program. DKNMDXR allows you to:

- Print selected fields from compressed and uncompressed codeline data records, in character, horizontal hexadecimal, or vertical hexadecimal formats.
- Print lines of text anywhere in the report.
- Print comments at the end of detail lines.

  Print multiple reports simultaneously as permitted by the DKNBLDL PRINT parameter for the application task.

Note: See “Mass Data Set” in the *CPCS-I Programming Guide* for information on codeline record types.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNMDXR:

Table 1-33. DKNMDXR Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNMDXR Call Parameters • Field PAGE-NUMBER may be used to modify the current page number. • Field RESERVED must be initialized to binary zeroes before issuing each ‘open report’ call, and left untouched for the remaining of the program’s execution. • Field CARR-CTRL may be used to manage vertical formatting of the report. • Field MSG-PAD is used to override the maximum number of lines per page default value (55). • Field FLAG is used to control each field’s processing. Each byte corresponds to each codeline record field in the order they are defined in the MDX macro. Field captions are also printed as defined in the MDX macro.	DKNRPP1C DKNRPP1X	DKNRPP1A
Application Task Control Block / Codeline Record Buffer • This control block is the application’s APTCB’ on ‘open report’ calls, and the codeline record buffer on other calls.	DKNCATCB DKNRPP2C	DKNAPTCB DKNRPP2A

DKNMDXR may return the following completion codes in the RETURN-CODE general register (COBOL programs) or register 15 (assembler programs):

Table 1-34. DKNMDXR Return Codes

Code (dec)	Description
+00	Successful completion.
+04	Report already open.
+08	Report not open.
+12	Parameter count error.
+16	Invalid function code.
+20	Dynamic load failure.
+24	Invalid field flag.
+32	Unavailable fixed storage.
+36	Unavailable variable storage.
+40	Bad DKNADCB2 return code.
+44	Bad OPEN return code.
+48	Variable storage size greater than 8K.
+52	Invalid fixed storage address.
+60	Text exceeds available storage.
+64	DKNBCFIO bad return code.

DKNMDXR may return combined returned codes. To determine the values of the two codes, the completion code must be first converted to hexadecimal. Digits 1–2 are the first return code. Digits 3–4 are the second return code.

Restrictions

To process multiple reports simultaneously, multiple copies of the communication control block must be used.

See “Describing an Application Task’s Environment” in the *CPCS-I Customization Guide* for a description of the DKNBLDL PRINT parameter.

Example 1

Operation:

Generate an uncompressed codeline data report, showing the account and amount fields, and the user code field (in horizontal hexadecimal format), and the comment `DOCUMENT TYPE 1` if the user code has the value E6, on each detail line. The lines per page default value is used. ‘REPORT 1’ is used as the report title and spool report description. Item count total lines follow each group of detail lines corresponding to a same account. A page break is forced after printing each total line.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSMDXR.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  REPORT1-DESCRIPTION        PIC  X(032) VALUE 'REPORT 1'.

 01  LAST-ACCOUNT-NUMBER        PIC  X(014).

 01  TEXT-LENGTH                PIC  9(002) VALUE 24.

 01  FIRST-RECORD-SW            PIC  X(001) VALUE 'Y'.
     88  FR-NOT-FIRST-RECORD                VALUE 'N'.

 01  ITEM-TOTALS.
     05  FILLER                 PIC  X(013) VALUE 'ACCOUNT ITEM'.
     05  FILLER                 PIC  X(006) VALUE 'COUNT'.
     05  IT-ITEM-COUNT          PIC  9(005) VALUE 0.

 01  DOCUMENT-TYPE1.
     05  FILLER                 PIC  X(009) VALUE 'DOCUMENT'.
     05  FILLER                 PIC  X(031) VALUE 'TYPE 1'.

 01  USER-CODE-TYPE1            PIC  X(001) VALUE X'E6'.

 01  PRINT1-HEADING1.
     05  FILLER                 PIC  X(043) VALUE ' '.
     05  FILLER                 PIC  X(008) VALUE 'REPORT 1'.

 01  REPORT1-COMPLETION-SW      PIC  X(001) VALUE 'N'.
     88  RC-REPORT1-COMPLETE                VALUE 'Y'.

 01  SUCCESSFUL-OPERATION       PIC  9(001) VALUE 0.

 COPY DKNCRZD                  REPLACING ==:ZD:==
                               BY        ==ZD==.

 COPY DKNRPP1C.

 COPY DKNRPP1X.

 COPY DKNRPP2C.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC  X(068).
/
 PROCEDURE DIVISION            USING APTCB
                                     START-PARMS.
```

Figure 1-45 (Part 1 of 3). DKNMDXR Example 1

```
     PERFORM 1000-OPEN-REPORT1.

     IF  RETURN-CODE                  =  SUCCESSFUL-OPERATION
         PERFORM 2000-CREATE-REPORT1
             UNTIL RC-REPORT1-COMPLETE
             OR    RETURN-CODE        NOT = SUCCESSFUL-OPERATION

         IF  RETURN-CODE              =  SUCCESSFUL-OPERATION
             SET  XR-1-F-CLOSE        TO TRUE
             CALL 'DKNMDXR'           USING XR-PARM-1
                                            DKNRPP2C.

     GOBACK.

 1000-OPEN-REPORT1.

     MOVE XR-IN-FLAG-DISP             TO XR-FLAG (XRMXFAMT)
                                        XR-FLAG (XRMXFACT)
                                        XR-FLAG (XRMXFTXT).

     MOVE XR-IN-FLAG-HHEX             TO XR-FLAG (XRMXFTPU).

     MOVE XR-IN-FLAG-NODISP           TO XR-FLAG (XRMXFPCT)
                                        XR-FLAG (XRMXFOP1)
                                        XR-FLAG (XRMXFRTR)
                                        XR-FLAG (XRMXFRTI)
                                        XR-FLAG (XRMXFSRN)
                                        XR-FLAG (XRMXFSEQ)
                                        XR-FLAG (XRMXF009)
                                        XR-FLAG (XRMXF010)
                                        XR-FLAG (XRMXF011)
                                        XR-FLAG (XRMXF012)
                                        XR-FLAG (XRMXF013)
                                        XR-FLAG (XRMXF014)
                                        XR-FLAG (XRMXF015)
                                        XR-FLAG (XRMXFRV1)
                                        XR-FLAG (XRMXFRV2)
                                        XR-FLAG (XRMXFRV3)
                                        XR-FLAG (XRMXFCPU)
                                        XR-FLAG (XRMXFLG1)
                                        XR-FLAG (XRMXFLG2)
                                        XR-FLAG (XRMXFTP1)
                                        XR-FLAG (XRMXFPKT)
                                        XR-FLAG (XRMXFLG3)
                                        XR-FLAG (XRMXFLG4)
                                        XR-FLAG (XRMXFUSR).

     SET  XR-1-F-OPEN                 TO TRUE.
     MOVE REPORT1-DESCRIPTION         TO XR-TITLE.
     MOVE PRINT1-HEADING1             TO XR-HDR-LINE-1.
     MOVE 0                           TO XR-RESERVED.

     CALL 'DKNMDXR'                   USING XR-PARM-1
                                      APTCB.
```

Figure 1-45 (Part 2 of 3). DKNMDXR Example 1

```
2000-CREATE-REPORT1.

    ...................
    ...................
    ...................

    IF  ...............
        SET  RC-REPORT1-COMPLETE     TO TRUE.

    ...................
    ...................
    ...................

    IF  FR-NOT-FIRST-RECORD
        IF  ZD-ACCOUNT-NUMBER        NOT = LAST-ACCOUNT-NUMBER
            SET  XR-1-F-PRINT-MSG    TO TRUE
            MOVE ITEM-TOTALS         TO MDXR-LINE-TEXT
            MOVE TEXT-LENGTH         TO MDXR-LINE-LENGTH
            MOVE 0                   TO IT-ITEM-COUNT

            CALL 'DKNMDXR'           USING XR-PARM-1
                                           DKNRPP2C

            MOVE XR-NEW-PAGE         TO XR-INP-CARR-CTRL.

    SET FR-NOT-FIRST-RECORD          TO TRUE.

    IF  RETURN-CODE                  =  SUCCESSFUL-OPERATION
        IF  ZD-USER-FLAGS            =  USER-CODE-TYPE1
            MOVE DOCUMENT-TYPE1      TO XR-COMMENT
        ELSE
            MOVE ' '                 TO XR-COMMENT
        END-IF

        MOVE ZD                      TO MDXR-MDS-UNC-UNE-REC
        SET  XR-1-F-PRINT-ZD         TO TRUE

        CALL 'DKNMDXR'               USING XR-PARM-1
                                           DKNRPP2C.

    IF  RETURN-CODE                  =  SUCCESSFUL-OPERATION
        ADD  1                       TO IT-ITEM-COUNT
        MOVE ZD-ACCOUNT-NUMBER       TO LAST-ACCOUNT-NUMBER
    ELSE
        SET  RC-REPORT1-COMPLETE     TO TRUE.
```

Figure 1-45 (Part 3 of 3). DKNMDXR Example 1

Example 2

Operation:

Generate a compressed codeline data report, showing the serial, account, flag 1, flag 2, flag 3, flag 4 and field 15 fields in vertical hexadecimal format. The lines per page default value is used. 'REPORT 1' is used as the report title and spool report description.

Assembler Code:

```
TSMDXR   CSECT
TSMDXR   AMODE 24
TSMDXR   RMODE 24
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE SUBROUTINE
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         MVC   XR_FLAGS,NOPRINT      SET FIELD SELECTION FLAGS:
         MVI   XR_FLAGS+XRMXFSRN-1,XRFLG_VHEX  SERIAL #,
         MVI   XR_FLAGS+XRMXFACT-1,XRFLG_VHEX  ACCOUNT #,
         MVI   XR_FLAGS+XRMXFLG1-1,XRFLG_VHEX  FLAG 1,
         MVI   XR_FLAGS+XRMXFLG2-1,XRFLG_VHEX  FLAG 2,
         MVI   XR_FLAGS+XRMXFLG3-1,XRFLG_VHEX  FLAG 3,
         MVI   XR_FLAGS+XRMXFLG4-1,XRFLG_VHEX  FLAG 4 AND
         MVI   XR_FLAGS+XRMXF015-1,XRFLG_VHEX  FIELD 15 IN VERTL HEX
         MVC   XR_TITLE,REP1DESC     SET SPOOL REPORT DESCRIPTION
         MVC   XR_HDR_LINE_1,REP1HDG1 SET REPORT HEADING 1
         LA    R4,0                  SET RESERVED
         ST    R4,XR_RESERVED          FIELD
         LA    R4,XRF_OPEN           SET OPEN
         STH   R4,XR_FUNCTION          FUNCTION
         MVC   PARM1,MDXRCP1@        GET DKNMDXR CALL PARMS 1 @
         MVC   PARM2,0(R1)           GET APTCB @
         OI    PARM2,ENDLIST         SET 'END OF LIST' FLAG
         LA    R1,PARMS              GET DKNMDXR PARM LIST @
         L     R15,MDXR@             GET DKNMDXR @
         BASSM R14,R15               CALL DKNMDXR TO OPEN THE RPRT
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BNZ   MDXRERR               N. PROCESS DKNMDXR ERROR
*
NEXTREC  DS   0H                     PROCESS NEXT RECORD
         .................
         .................
         .................
         B     PRINTREC              PRINT RECORD
*
         .................
         .................
         .................
         B     REPEND                ALL RECORDS PROCESSED
*
PRINTREC MVC   DKNRPP2A,.....        SET CODELINE DATA RECORD
         LA    R4,XRF_PRT_ZC         SET PRINT UNCOMPRESSED
         STH   R4,XR_FUNCTION          FUNCTION
         MVC   PARM1,MDXRCP1@        GET DKNMDXR CALL PARMS 1 @
         MVC   PARM2,MDXRCP2@        GET DKNMDXR CALL PARMS 2 @
         OI    PARM2,ENDLIST         SET 'END OF LIST' FLAG
```

Figure 1-46 (Part 1 of 2). DKNMDXR Example 2

```
         LA    R1,PARMS              GET DKNMDXR PARM LIST @
         L     R15,MDXR@             GET DKNMDXR @
         BASSM R14,R15               CALL DKNMDXR TO PRINT A RECORD
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BNZ   MDXRERR               N. PROCESS DKNMDXR ERROR
         B     NEXTREC               PROCESS NEXT RECORD
*
REPEND   LA    R4,XRF_CLOSE          SET CLOSE
         STH   R4,XR_FUNCTION          FUNCTION
         LA    R1,PARMS              GET DKNMDXR PARM LIST @
         L     R15,MDXR@             GET DKNMDXR @
         BASSM R14,R15               CALL DKNMDXR TO CLOSE THE RPRT
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BZ    RETURN                Y. EXIT PROGRAM
*
MDXRERR  DS   0H                     PROCESS DKNMDXR ERROR
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
REP1HDG1 DS   0H                     REPORT HEADING 1
         DC  34C' '                  . FILLER
         DC    C'REPORT 1'           . TITLE
         DC  80C' '                  . FILLER
*
MDXR@    DC    V(DKNMDXR)            DKNMDXR @
MDXRCP1@ DC    A(XR_FUNCTION)        DKNMDXR CALL PARMS 1 @
MDXRCP2@ DC    A(DKNRPP2A)           DKNMDXR CALL PARMS 2 @
*
ENDLIST  EQU   X'80'                 'END OF LIST' FLAG
*
NOPRINT  DC  28C'N'                  FLAG INITIALIZATION (NO PRINT)
*
PARMS    DS   0F                     DKNMDXR OPEN PARM LIST
PARM1    DS    A                     . CALL PARMS
PARM2    DS    A                     . APTCB @ (OPEN)/CODELINE DATA
*
REP1DESC DC    CL32'REPORT 1'        REPORT DESCRIPTION
*
         COPY  DKNRPP1A              DKNMDXR CALL PARMS
*
         COPY  DKNRPP2A              DKNMDXR CALL CODELINE DATA AREA
*
SAVEAREA DS  18F                     OUR SAVE AREA
*
         END
```

Figure 1-46 (Part 2 of 2). DKNMDXR Example 2

Performing Security Checks

CPCS-I application tasks may perform transaction level security verifications, independently of the security type being used, by calling module DKNMAYI.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNMAYI:

Table 1-35. DKNMAYI Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB

Transaction Name

- This is a 44-character field that contains the name of the transaction requiring security verification. The name must be left-justified and padded with blanks.

DKNMAYI may return the following completion codes in the return-code special register (COBOL programs) or register 15 (assembler programs):

Table 1-36. DKNMAYI Return Codes

Code (hex)	Description
+00	The transaction is authorized.
All others	The transaction is not authorized.

Example 1

Operation:

Create a task that performs transaction TRAN1 if it is authorized, and reports the authorization failure otherwise.

COBOL Code:

```
ID DIVISION.
PROGRAM-ID. TSMAYI.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

Figure 1-47 (Part 1 of 2). DKNMAYI Example 1

```
 01  TRANSACTION-ID             PIC  X(44) VALUE 'TRAN1'.
 01  AUTHORIZED-TRANSACTION     PIC  9(01) VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01   START-PARMS               PIC  X(68).
/
 PROCEDURE DIVISION                  USING APTCB
                                           START-PARMS.

     ...................
     ...................
     ...................

     CALL 'DKNMAYI'                  USING APTCB
                                           TRANSACTION-ID.

     IF  RETURN-CODE                 = AUTHORIZED-TRANSACTION
         PERFORM 1000-PROCESS-TRANSACTION
     ELSE
         PERFORM 2000-REPORT-AUTHORIZATION-FAILURE.

     ...................
     ...................
     ...................

     GOBACK.

 1000-PROCESS-TRANSACTION.

     ...................
     ...................
     ...................

 2000-REPORT-AUTHORIZATION-FAILURE.

     ...................
     ...................
     ...................
```

Figure 1-47 (Part 2 of 2). DKNMAYI Example 1

Example 2

Operation:

Create a task that performs transaction TRAN2 if it is authorized, and reports the authorization failure otherwise.

Assembler Code:

```
TSMAYI   CSECT
TSMAYI   AMODE 31
TSMAYI   RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
         .................
         .................
         .................
*
         LA    R1,PARMS             GET SECURITY INTERFACE PARMS @
         L     R15,MAYI@            GET SECURITY INTERFACE @
         BASSM R14,R15              CALL SECURITY INTERFACE @
         LTR   R15,R15              TRANSACTION AUTHORIZED ?
         BNE   NOTAUTH              N. REPORT AUTHORIZATION FAILURE
*
AUTH     DS   0H                    PERFORM TRANSACTION
         .................
         .................
         .................
*
NOTAUTH  DS   0H                    REPORT AUTHORIZATION FAILURE
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
MAYI@    DC    V(DKNMAYI)           SECURITY INTERFACE @
*
PARMS    DS   0F                    DKNMAYI PARM LIST
APTCB@   DS    A                   . APTCB @
TRANID@  DC    A(TRANID)           . TRANSACTION ID @
*
TRANID   DC    CL44'TRAN2'         TRANSACTION ID
*
         END
```

Figure 1-48. DKNMAYI Example 2

Creating DFTI Input Data Sets

CPCS-I tasks may call routine DKNCDIF to unload one or more strings into a sequential data set to be used as input to the DFTI task.

The calling task must pass the name of a temporary sequential data set that contains the names of the strings to be unloaded.

The output sequential data set contains a header record, multiple detail codeline records in either compressed or uncompressed format, and a trailer record, in the format expected by DFTI. The data set may be either temporary or permanent. If temporary, the DSName is the DSName assigned to the ddname CDIF, in the DKNDSAT table, with the Thhmmsst extension, where hhmmsst is the allocation time (see “DKNTDYNA — Dynamically Allocate/Unallocate Temporary Data Sets” on page 4-33).

DKNCDIF returns the total number of records written to the output data set.

DKNCDIF calls an optional user exit routine before each write to the output data set (see “CDIF Exit — Modify DEFT Input Data Set Creation” on page 2-7).

Messages CDIF 30005, 00001 and 00002, indicating that DFTI must be run, and showing the output data set name, are displayed on the supervisor terminal and sent to the scroll log when DKNCDIF completes successfully.

Note: The CPCS-I task OLMS calls DKNCDIF.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNCDIF:

Table 1-37. DKNCDIF Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
DKNCDIF Call Parameters • The original and changed ddname fields must contain the DKNQPUT returned information about the temporary string name data set. • The permanent ddname field must contain the ddname of the output data set (if it is a permanently allocated data set), or blanks (DKNCDIF uses the CDIF ddname information in the DKNDSAT table to allocate a temporary data set then).	DKNCRCDP	DKNCDP

COBOL programs may use copybook DKNCRCDI to reference the records in the string name data set passed to DKNCDIF.

DKNCDIF may return the following completion codes in the call parameters:

Table 1-38. DKNCDIF Return Codes

Code (dec)	Description
+00	Successful Completion
+02	Codeline Record I/O Error
+04	Data Set Retrieval Error
+08	Data Set Generation Error
+12	User Exit Routine Requested Termination
+16	DKNDEFTD Data Set Access Error

Messages CDIF 39018 and 30003, indicating the type of error, are displayed on the supervisor terminal and sent to the scroll log, if DKNCDIF does not complete successfully.

Restrictions

The calling program is responsible for creating the temporary string name data set via the DKNQPUT (see “DKNQPUT — Perform QSAM Output Processing” on page 4-28) subroutine.

Example 1

Unload strings 0004-1-00-00-00-00-I-000 and 0005-1-00-00-00-00-I-000 into a temporary sequential data set to be used as input to DFTI. Codeline records are written in uncompressed format.

COBOL Code:

```
  ID DIVISION.
  PROGRAM-ID. TSCDIF.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  ORIGINAL-DDNAME                  PIC X(08) VALUE 'TEMP'.

  01  STRING-NAMES.
      05  SN-STRING-NAME-1.
          10  FILLER                   PIC X(04) VALUE '0004'.
          10  FILLER                   PIC X(01) VALUE '1'.
          10  FILLER                   PIC X(08) VALUE '00000000'.
          10  FILLER                   PIC X(01) VALUE 'I'.
          10  FILLER                   PIC X(03) VALUE '000'.
```

Figure 1-49 (Part 1 of 3). DKNCDIF Example 1

```
      05  SN-STRING-NAME-2.
          10  FILLER                  PIC X(04) VALUE '0005'.
          10  FILLER                  PIC X(01) VALUE '1'.
          10  FILLER                  PIC X(08) VALUE '00000000'.
          10  FILLER                  PIC X(01) VALUE 'I'.
          10  FILLER                  PIC X(03) VALUE '000'.

  COPY DKNQPUTC                        REPLACING ==:TAG:==
                                       BY        ==TEMP==.

  COPY DKNCRCDI.

  COPY DKNCRCDP.
 /
  LINKAGE SECTION.

  COPY DKNCATCB.

  01  START-PARMS                     PIC X(68).
 /
  PROCEDURE DIVISION                   USING APTCB
                                             START-PARMS.

      PERFORM 1000-CREATE-STRING-NAME-FILE.
      MOVE QPUT-TEMP-STATUS-AREA       TO RETURN-CODE.

      IF  QPUT-TEMP-GOOD
          PERFORM 2000-CREATE-DFTI-INPUT-FILE.
          MOVE CDIFP-RETURN-CODE       TO RETURN-CODE.

      GOBACK.

  1000-CREATE-STRING-NAME-FILE.

      SET  QPUT-TEMP-PUT-TEMP          TO TRUE.
      MOVE ORIGINAL-DDNAME             TO QPUT-TEMP-ORIG-DDNAME.
      MOVE SN-STRING-NAME-1            TO CDIFI-STRING-NAME.
      PERFORM 1500-CALL-QPUT.

      IF  QPUT-TEMP-GOOD
          MOVE SN-STRING-NAME-2        TO CDIFI-STRING-NAME
          PERFORM 1500-CALL-QPUT

          IF  QPUT-TEMP-GOOD
              SET  QPUT-TEMP-CLOSE     TO TRUE
                   PERFORM 1500-CALL-QPUT.

  1500-CALL-QPUT.

      CALL 'DKNQPUT'                   USING QPUT-TEMP-PARM-LIST
                                             CDIFI-PARMS
                                             APTCB.

  2000-CREATE-DFTI-INPUT-FILE.
```

Figure 1-49 (Part 2 of 3). DKNCDIF Example 1

```
      SET  CDIFP-DEFTI-DI             TO TRUE.
      MOVE ' '                        TO CDIFP-PERM-DDNAME.
      MOVE ORIGINAL-DDNAME            TO CDIFP-ORIG-QPUT-DDNAME.
      MOVE QPUT-TEMP-CHGD-DDNAME      TO CDIFP-CHGD-QPUT-DDNAME.

      CALL 'DKNCDIF'                  USING APTCB
                                            CDIFP-PARMS.
```

Figure 1-49 (Part 3 of 3). DKNCDIF Example 1

Example 2

Unload string 0004-1-00-00-00-00-I-000 into a permanent sequential data set allocated to the ddname PERM for DFTI processing. Codeline records are written in compressed format.

Assembler Code:

```
TSCDIF   CSECT
TSCDIF   AMODE 31
TSCDIF   RMODE ANY
*
         SAVE  (14,12)             SAVE REGISTERS
         BASR  R12,0               GET ROUTINE @
         USING *,R12               BASE SUBROUTINE
*
         LA    R14,SAVEAREA        POINT TO OUR SAVE AREA
         ST    R14,8(R13)          STORE FORWARD SAVE AREA @
         ST    R13,4(R14)          STORE BACKWARD SAVE AREA @
         LR    R13,R14             POINT TO NEW SAVE AREA
*
         MVC   QPAPTCB@,0(R1)      SAVE APTCB @ INTO DKNQPUT PARMS
         MVC   CDAPTCB@,0(R1)      SAVE APTCB @ INTO DKNCDIF PARMS
*
         MVC   QPODDN,ORDDN        SET ORIGINAL DDNAME
         XC    QPRSV1,QPRSV1       INITIALIZE RESERVED AREA 1
         XC    QPRSV2,QPRSV2       INITIALIZE RESERVED AREA 2
         MVC   QPREQ,PUTTEMP       SET 'WRITE TO TEMPORARY' REQUEST
         LA    R1,QPUTPRMS         GET QSAM PUT PARM LIST @
         L     R15,QPUT@           GET QSAM PUT @
         BASSM R14,R15             CALL QSAM PUT TO WRITE RECORD
         CLC   QPSTAT,QPUTDONE     RECORD WRITTEN ?
         BNE   QPUTER              N. PROCESS ERROR
*
         MVC   QPREQ,CLOSE         SET 'CLOSE' REQUEST
         LA    R1,QPUTPRMS         GET QSAM PUT PARM LIST @
         L     R15,QPUT@           GET QSAM PUT @
         BASSM R14,R15             CALL QSAM PUT TO CLOSE DATA SET
         CLC   QPSTAT,QPUTDONE     DATA SET CLOSED ?
         BNE   QPUTER              N. PROCESS ERROR
*
```

Figure 1-50 (Part 1 of 3). DKNCDIF Example 2

```
         MVC   CLFORMAT,COMP        SET 'COMPRESSED FORMAT'
         MVC   CLPRNAME,OUTDD       SET OUTPUT DATA SET DDNAME
         MVC   CLORIGDD,ORDDN       SET STRING NAME ORIGINAL DDNAME
         MVC   CLCHGDDD,QPCDDN      SET STRING NAME CHANGED DDNAME
         LA    R1,CDIFPRMS          GET DKNCDIF PARM LIST @
         L     R15,CDIF@            GET DKNCDIF @
         BASSM R14,R15              CALL DKNCDIF TO CREATE DFTI INPUT
         CLC   CLRETCDE,CDIFDONE    SUCCESSFUL OPERATION ?
         BE    RETURN               Y. EXIT PROGRAM
*
CDIFER   DS   0H                    PROCESS DKNCDIF ERROR
         .............
         .............
         .............
         B     RETURN               EXIT PROGRAM
*
QPUTER   DS   0H                    PROCESS DKNQPUT ERROR
         .............
         .............
         .............
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
COMP     DC    C'ZD'                COMPRESSED RECORD FORMAT REQUEST
*
         DS   0H                    DKNQPUT REQUEST CODES
PUTTEMP  DC    C'PT'                . PUT
CLOSE    DC    C'C '                . CLOSE
*
         DS   0H                    REQUEST CODES
QPUTDONE DC    H'0'                 . DKNQPUT REQUEST COMPLETED
CDIFDONE DC    C'0000'              . DKNCDIF REQUEST COMPLETED
*
ORDDN    DC    CL8'TEMP'             STRING NAME ORIGINAL DDNAME
OUTDD    DC    CL8'PERMW'            OUTPUT DATA SET DSNAME
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
QPUT@    DC    V(DKNQPUT)           QSAM PUT @
CDIF@    DC    V(DKNCDIF)           DFTI INPUT CREATE @
*
QPUTPRMS DS   0F                    DKNQPUT PARM LIST
         DC    A(QPPLIST)           . CALL PARMS @
         DC    A(STGNAME)           . I/O AREA @
QPAPTCB@ DS    A                    . APTCB @
*
CDIFPRMS DS   0F                    DKNCDIF PARM LIST
CDAPTCB@ DS    A                    . APTCB @
         DC    A(CDIFLIST)          . CALL PARMS
*
STGNAME  DC    CL80'00041000000000I000' DKNQPUT I/O AREA
*
```

Figure 1-50 (Part 2 of 3). DKNCDIF Example 2

```
         COPY  DKNQPUTA             DKNPUTC CALL PARAMETERS
*
         COPY  DKNCDP               DKNCDIF CALL PARAMETERS
*
         END
```

Figure 1-50 (Part 3 of 3). DKNCDIF Example 2

Interfacing with Host Codeline Edit Routines

CPCS-I tasks may call routine DKNMOLRI to interface with a codeline record host edit routine.

The calling task may request the following functions:

1. Open – Load host edit routine and optional table (first call).
2. Process – Validate codeline (one call per record).
3. Close – Delete host edit routine and optional table (final call).

An exit routine may optionally be called by DKNMOLRI on the 'open' and 'close' functions (see "MOLRI Exit — Modify Host's Edit Routine and Table Load Process" on page 2-88, for information on the MOLRI exit).

DKNMOLRI uses the edit routine and optional table names specified in the MUPA control block (see "Linkage") during the 'open' process. If the MUPA fields contain blanks or low-values, the host edit routine and table names are retrieved from the passed string-header record (the edit routine's name last character is changed to 'R').

Note: CPCS-I tasks OLRR, ADJ and DFTI call the DKNMOLRI interface.

DKNMOLRI sends messages MOLRI00001 and MOLRI00002 to the scroll log when its 'open' and 'close' functions complete successfully. Part of the information included in these messages (sort type, cycle ID, entry number and pass pocket history) is present only when the calling task passes that information to DKNMOLRI in the MUPA control block.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks.

Table 1-39. DKNMOLRI Communication Control Blocks

Control Block	Assembler Macro/ Copybook
Application Task Control Block	DKNAPTCB
MICR/MOLRI User Parameter Area • Field MUPAFUNC contains the function code. Fields MUPASSN and MUPASTN may contain the edit routine name and table name, respectively, on 'open' and 'close' calls.	MUPA
Codeline Record Buffer • This control block is required. If it does not contain the codeline record to be edited, the MUPA area MUPAIMGA field must contain the codeline record address.	MRDBDSCT

DKNMOLRI may return the following completion codes in register 15:

Table 1-40. DKNMOLRI Routine Return Codes

Code (dec)	Description
+00	DKNMOLRI processed successfully.
+04	DKNMOLRI could not allocate storage for its save and work areas.
+08	DKNMOLRI could not load a service routine, the exit routine, the host codeline edit routine, or the table (if applicable).
+12	DKNMOLRI did not receive a non-blank edit routine name or a standard string-header record on the 'open' function.
+16	DKNMOLRI received a non-zero return code from the control-document determination routine (DKNTYPER).
+20	The MOLRI exit routine requested the task's termination.
+24	The passed MUPA control block parameters were not valid. Scroll log messages indicate the fields causing the error.
+28	DKNMOLRI could not access the Bank Control Data Set.
+32	DKNMOLRI received an incorrect return code from the host edit routine.
+36	The string-header record did not contain a valid, non-blank, host codeline edit routine name.

DKNMOLRI sends an error message to the scroll log when it returns with a non-zero completion code.

Restrictions

The calling program must be written in assembler.

The MUPA control block MUPAMLWA field must contain low-values on the first DKNMOLRI call.

DKNMOLRI does not pass tracer and divider codeline records to the edit routine unless the MUPA control block MUPAFLG2 field is equated to the MUPAPASA flag.

The sort type, cycle ID, entry number and pass pocket history must be included in the MUPA control block.

Example 1

Re-assign pocket numbers to the records contained in a string, editing them with the host edit routine specified in the string header record. Tracer and divider codeline records are not edited.

Assembler Code:

```
TSMOLRI  CSECT
TSMOLRI  AMODE 31
TSMOLRI  RMODE ANY
*
         SAVE  (14,12)             SAVE REGISTERS
         BASR  R12,0               GET ROUTINE @
         USING *,R12               BASE SUBROUTINE
*
         LA    R14,SAVEAREA        POINT TO OUR SAVE AREA
         ST    R14,8(R13)          STORE FORWARD SAVE AREA @
         ST    R13,4(R14)          STORE BACKWARD SAVE AREA @
         LR    R13,R14             POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)        GET APTCB @
         ..............
         ..............
         ..............
*
NEXTREC  DS   0H                   EDIT CODELINE RECORD
         ..............
         ..............
         ..............
         B     CLOSCALL            ALL RECORDS PROCESSED
*
         ..............
         ..............
         ..............
         MVC   MRDIMG,DI           SET COMPRESSED CODELINE RECORD
         TM    DIFLAG2,DIMTCR      CONTROL RECORD ?
         BZ    SETPROC             N. REQUEST 'PROCESS' FUNCTION
         CLI   DITYPEI,DISTGHD     STRING HEADER RECORD ?
         BNE   SETPROC             N. REQUEST 'PROCESS' FUNCTION
         MVI   MUPAFUNC,MUPAOPEN   SET 'OPEN' FUNCTION
         B     CLLMOLRI            SKIP 'PROCESS' FUNCTION REQUEST
*
SETPROC  MVI   MUPAFUNC,MUPAEDIT   SET 'PROCESS' FUNCTION REQUEST
*
CLLMOLRI LA    R1,PARMS            GET DKNMILRI PARM LIST @
         L     R15,MOLRI@          GET DKNMOLRI @
         BASSM R14,R15             CALL DKNMOLRI
         LTR   R15,R15             SUCCESSFUL OPERATION ?
         BNZ   RETURN              N. EXIT WITH DKNMOLRI RETURN CODE
*
PROCREC  DS   0H                   PROCESS EDITED CODELINE RECORD
         ..............
         ..............
         ..............
         B     NEXTREC             EDIT NEXT STRING CODELINE RECORD
*
CLOSCALL MVI   MUPAFUNC,MUPACLOS   SET 'CLOSE' FUNCTION
         LA    R1,PARMS            GET DKNMOLRI PARM LIST @
         L     R15,MOLRI@          GET DKNMOLRI @
         BASSM R14,R15             CALL DKNMOLRI
*
```

Figure 1-51 (Part 1 of 2). DKNMOLRI Example 1

```
RETURN   L     R13,SAVEAREA+4      RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)  RETURN
*
         DKNREGS                   REGISTER EQUATES
*
MOLRI@   DC    V(DKNMOLRI)         DKNMOLRI @
*
PARMS    DS   0F                   DKNMOLRI PARM LIST
APTCB@   DS    A                   . APTCB @
         DC    A(MUPA)             . MUPA @
         DC    A(MRDBUF)           . MRDBUF @
*
         COPY  MUPA                DKNMOLRI USER PARAMETER AREA
*
         COPY  MRDBDSCT            CODELINE RECORD BUFFER
*
SAVEAREA DS  18F                   OUR SAVE AREA
*
         END
```

Figure 1-51 (Part 2 of 2). DKNMOLRI Example 1

Invoking User Exits With the User Exit Manager

The CPCS-I user exit facility is designed to isolate CPCS-I programs from user exit processing. The user exit facility is invoked by calling module DKNUEM. To use this facility, a CPCS-I program simply calls DKNUEM at a defined exit point, and DKNUEM does the rest:

- Calls all exits defined for that exit point (up to the user-defined maximum number of exits for each exit point).
- Establishes abend protection so that user exit abends do not cause the calling program to abend.
- Returns control to the calling program after one of the following occurs:
  - All exits called with return codes of zero
  - Non-zero return code from a user exit
  - Calling program requested control after each user exit
  - User Exit Manager encountered an error condition

Activation

The CPCS-I User Exit Facility is activated during CPCS-I system initialization and remains active until CPCS-I system termination. Field UEM@, in the CPCS-I parameter list control block, contains the DKNUEM load address.

Linkage

Table 1-41. DKNUEM Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
CPCS-I Parameter List Address	DKNCPARM	DKNPARM
User Exit Request Block Address	DKNUERQ2	DKNUERQ1

User Exit Parameter List Address
(address of the parameter list for the user exit)

For all requests, the User Exit Point Name must be specified in the User Exit Request Block.

If the calling program is requesting either a RETRY or a RESUME request, a “Y” must be specified in the appropriate field in the User Exit Request Block.

When DKNUEM returns control to the calling program, the following information is available:

Cobol Field	Assembler Field	Description
N/A	R15/R0	Return/reason codes, respectively
UER-RETURN-CODE	UERRC	Return code (if -1, it is a DKNUEM error)
UER-REASON-CODE	UERREA	Reason code (if UERRC = -1, it is a DKNUEM reason)
UER-CURRENT-EXIT-NAME	UERCEXIT	Current exit name
UER-CURRENT-EXIT- SCC	UERSCC	Current system abend code if DKNUEM or user exit abended
UER-CURRENT-EXIT- UCC	UERUCC	Current user abend code if DKNUEM or user exit abended
UER-PROCESSING-COMPLETE	UERDONE	All processing complete indication (“Y” = all processing complete, “N” = all exits have not been called.

DKNUEM calls all exits for that exit point and gives control back to the calling program. If there is a problem, DKNUEM gives control back to the calling program with a special return code of -1 and a reason code detailing the nature of the error. The list of reason codes and their meanings are as follows:

Table 1-42. DKNUEM Return/Reason Codes

Return Code	Reason Code	Description
-1	4 (X'04')	The exit point was not found. This means that no user exits were specified for this exit point.
-1	8 (X'08')	The exit point is not active. An authorized operator has deactivated this exit point.
-1	12 (X'0C'	No exits exist for this exit point. An exit point identifier was specified but no exit was identified for it.
-1	16 (X'10')	The user exit is not active. The current user exit has been deactivated by an authorized operator.
-1	20 (X'14')	Error loading the user exit. An error occurred during an MVS LOAD macro invocation.
-1	24 (X'18')	User exit abended. The current user exit abended.
-1	28 (X'1C')	User Exit Manager abended. The DKNUEM module abended.

Restrictions

The calling program is responsible for handling each of the DKNUEM exception conditions. To help with this task, DKNUEM allows the calling program to retry the current exit (after an abend or non-zero return code) by turning on the RETRY indicator in the DKNUEM communication control block and by calling DKNUEM again. Also, the calling program may wish to skip the current user exit and continue processing. The RESUME indicator is used for this purpose.

When all user exit processing is complete for this exit point, the DONE indicator is turned on, indicating this condition.

Example

```
*
*---------------------------------------------------------------------*
*                        Call the Exit Handler                        *
*---------------------------------------------------------------------*
*
MAIN10   DS    0H
         CLI   UERDONE,C'Y'            Have all exits been called?
         BE    MAIN98                  Yes...We are outta here...
         MVC   UERUEP(32),=CL32'TEST_EXIT_POINT'
*                                       Specify Exit Point Name
         MVI   UEREVERY,C'Y'           I want control after each exit!
*comment MVI   UEREVERY,C'N'           Don't come back until finished!
         MVC   UXITPRM@(4),INPUT@      Specify User Exit Parmlist Addr
         MVC   PARMLST@(4),TCBPARM     Pass CPCS-I Parameter List Address
         LA    R1,DKNUERQ1             Get Request Block Address
         ST    R1,UEREQ@               Pass it to DKNUEM
         LA    R1,UEMPRM               Get the Input Parmlist Address
         L     R15,UEM@                Get User Exit Manager Address
         BASSM R14,R15                 Call the Exit Handler
         LTR   R15,R15                 Were all exits successful?
         BZ    MAIN97                  Yes....We are finished...
         C     R15,=F'-1'              This a User Exit Facility error?
         BNE   MAIN96                  No...a User Exit had a problem..
*
*---------------------------------------------------------------------*
*                    User Exit Facility Error!!!!!!!                  *
*---------------------------------------------------------------------*
*
         C     R0,=F'4'                Exit Point Not Found?
         BE    MAIN95                  Yes...Produce Error Message
         C     R0,=F'8'                Exit Point Deactivated?
         BE    MAIN94                  Yes...Produce Error Message
         C     R0,=F'16'               User Exit Deactivated?
         BE    MAIN93                  Yes...Produce Error Message
         C     R0,=F'20'               User Exit LOAD Error...
         BE    MAIN92                  Yes...Produce Error Message
         C     R0,=F'24'               User Exit Abended...
         BE    MAIN91                  Yes...Produce Error Message
         C     R0,=F'28'               User Exit Abended...
         BE    MAIN90                  Yes...Produce Error Message
```

Figure 1-52. DKNUEM Example

Chapter 2. CPCS-I User Exit Programming

Overview 2-3
AllO Exit — Perform Document Processor Allocation/Unallocation Request Additional Validations 2-4
CDIF Exit — Modify DEFT Input Data Set Creation 2-7
CSBU Exits 1/2 — Modify Edit Routine and Table Load Processing 2-11
CYCL Exit — Validate Cycle and Endorse Dates 2-12
DFTI Exit 1 — Validate DEFT Input Initialization 2-15
DFTI Exit 2 — Modify DEFT Input Processing 2-19
DFTO Exit — Modify DEFT Output Processing 2-23
DFTP Exit — Analyze DFTP Activity 2-29
FSCN Exit — Modify FSCN Processing 2-33
HCDM Exit 1 — Modify HCDM Profile Parameters 2-37
HCDM Exit 2 — Modify HCDM Reconciliation String Creation 2-39
HCDM Exit 3 — Modify HCDM Unmatched Item Processing 2-44
HCDM Exit 4 — Modify HCDM Positional Match Sort Criteria 2-46
IDCR Exit — Access Adjustment Descriptions 2-48
MDIS Exit — Modify M-string Distribution 2-50
MICR Exit 1 — Customize MICR Entry Begin Parameter Validations 2-54
MICR Exit 2 — Customize MICR Document Processing 2-59
MICR Exit 3 — Customize MICR Codeline Data Match Processing 2-69
MICR Exit 4 — Customize Document Processor's Edit Routine and Table Load Processing (Non-XF Sort Types) 2-77
MICR Exits 5/6 — Customize Document Processor's Edit Routine and Table Load Processing (XF Sort Types) 2-83
MOLRI Exit — Modify Host's Edit Routine and Table Load Process 2-88
MRGE Exit — Modify HSRR MRGE Processing 2-93
MTASK Exit 1 — Perform Additional CPCS-I Startup Processing 2-95
MTASK Exit 2 — Perform Additional CPCS-I Shutdown Processing 2-97
ODCR Exit — Access Adjustment Descriptions 2-99
RCVY Exit — Modify Recovery Processing 2-100
SECR Exit — Perform User Security Validations 2-102
VSMGR Exit 1 — Perform CPCS-I Startup VSAM Data Set Processing . . 2-107
VSMGR Exit 2 — Perform CPCS-I Shutdown VSAM Data Set Processing 2-110
VTASK Exit 1 — VSCRL Special Buffer Processing 2-112
VTASK Exit 2 — VSCRL Special Scroll Initialization Processing 2-113
VTASK Exits 3/4 — VCOMS Special PCB Release Processing 2-114
VTASK Exit 5 — VEXTS Special Terminal Logon Processing 2-115

Overview

This chapter documents the user exits provided by CPCS-I. These exits can be used to modify the default process performed by some CPCS-I tasks. You are responsible for creating the appropriate user exit routines according to the requirements of your financial institution.

The *CPCS-I Programming Guide* documents the CPCS-I modules owning the user exits described in this chapter.

See the copybooks for the user exit communications control blocks.

* Trademark of IBM

AllO Exit — Perform Document Processor Allocation/Unallocation Request Additional Validations

This exit allows the further validation of the operator-entered request immediately after the CPCS-I validations were successfully performed and before the actual document processor allocation/unallocation operation takes place.

Activation

ALLO user exit routines are activated through the CPCS-I User Exit Facility. Use the following exit-point name (see the "DKNPEXIT-User Exit Profile Member" section in the CPCS-I Customization Guide) to activate this user exit.

```
DKNALLO_REQUEST_VALIDATION=xxxxxxxx
```

xxxxxxxx is the ALLO user exit routine name.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following control blocks:

Table 2-1. ALLO Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters	DKNCALL1	DKNCALL2

The exit routine gives a return a code indicating whether the requested operation is valid or invalid. An error message is returned when the user exit routine identifies the request as invalid.

Example 1

Operation:

Ensure that you use valid document processor numbers in two processing centers. Each center uses its own CPCS-I system with a unique auto-restart ID (A and B) and the same document processor definitions. Document processors 10-19 are reserved for center A. Document processors 20-29 are reserved for center B. The error message `'SORTER nn INVALID FOR CENTER x'` is returned by the user exit routine when an invalid document processor number is specified in the allocation request.

User Exit Routine:

```
ALLOEXT1 CSECT
ALLOEXT1 AMODE 31
ALLOEXT1 RMODE 24
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         GET OUR SAVE AREA @
         ST    R14,8(R13)           STORE FORWRD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWD SAVE AREA @
         LR    R13,R14              GET NEW SAVE AREA @
*
         L     R2,0(R1)             GET INTERFACE CNTL BLK @
         USING AEINCNTL,R2          MAP INTERFACE CNTL BLOCK
*
         CLI   AEOPERRQ,AEALLOCT    ALLOCATION REQUST?
         BNE   VALREQ               N. VALID REQUEST
*
         L     R3,AEAPTCB@          GET ALLO'S APTCB @
         USING APTCB,R3             MAP ALLO'S APTCB
         L     R3,TCBPARM           GET CPCSI PARM LIST @
         DROP  R3                   RELEASE REGISTER
         USING PARMLST,R3           MAP CPCSI PARM LIST
*
CHKCTRA  CLC   AUTORST,CENTERA      CENTER A ?
         BNE   CHKCTRB              N. CHECK FOR NEXT CENTER
         CLC   AESTRNBR,CTRALOW     STR # < LOW RANGE?
         BL    INVALREQ             Y. INVALID REQUEST
         CLC   AESTRNBR,CTRAHIGH    STR # > HIGH RNGE ?
         BH    INVALREQ             Y. INVALID REQUEST
         B     VALREQ               VALID REQUEST
*
CHKCTRB  CLC   AUTORST,CENTERB      CENTER B ?
         BNE   VALREQ               N. VALID REQUEST
         CLC   AESTRNBR,CTRBLOW     STR # < LOW RANGE ?
         BL    INVALREQ             Y. INVALID REQUEST
         CLC   AESTRNBR,CTRBHIGH    SRTR # > HIGH RNG?
         BH    INVALREQ             Y. INVALID REQUEST
*
VALREQ   MVI   AERETCDE,AEVALREQ    SET 'VALID' RET CD
         B     RETURN               EXIT ROUTINE
*
INVALREQ MVC   EMSORTER,AESTRNBR    STR # TO ERROR MSG
         MVC   EMCENTER,AUTORST     CENTR ID TO ERR MSG
         MVC   AEERRMSG,ERRMSG      SET ERROR MESSAGE
         MVI   AERETCDE,AEINVREQ    SET 'INVAL' RET CD
*
RETURN   L     R13,SAVEAREA+4       REST CLLR'S SV AREA @
         XRETURN (14,12)            RETURN TO ALLO
*
```

Figure 2-1 (Part 1 of 2). CDIF Exit Routine Example 1

```
         DKNREGS                      REGISTER EQUATES
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
CENTERA  DC    C'A'                   CENTER A'S CPCSI JOB ID
CENTERB  DC    C'B'                   CENTER B'S CPCSI JOB ID
*
CTRALOW  DC    C'10'                  CENTER A'S STR LOW RANGE
CTRAHIGH DC    C'19'                  CENTER A'S STR HIGH RNGE
*
CTRBLOW  DC    C'20'                  CENTER B'S STR LOW RANGE
CTRBHIGH DC    C'29'                  CENTER B'S STR HIGH RNGE
*
ERRMSG   DS   0CL52                   ERROR MESSAGE
         DC    C'SORTER '
EMSORTER DC    C'NN'
         DC    C' INVALID FOR CENTER '
EMCENTER DC    C'X'
         DC  22C' '
*
         DKNCALL1 DSECT=YES           ALLO INTERFACE CNTL BLK
*
         COPY  DKNAPTCB               APTCB
*
         COPY  DKNPARM                CPCSI PARM LIST
*
         END
```

Figure 2-1 (Part 2 of 2). CDIF Exit Routine Example 1

CDIF Exit — Modify DEFT Input Data Set Creation

The CDIF user exit routine allows you to:

- Exclude records from being written to the CDIF output data set.
- Stop further CDIF processing.

The CDIF exit routine receives each record before it is written to the output data set.

Note: CDIF is not a CPCS-I task. DKNCDIF is a subroutine called to convert strings to DFTI input format (see “Creating DFTI Input Data Sets” on page 1-123).

Activation

A different CDIF exit routine may be specified for each financial institution. The exit is active when one or more CDIF exit routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information.

The CDIF exit also is active when the exit routine name is specified in the DKNCDIF call-parameters. The call-parameters activation takes precedence over the bank control file specifications.

No re-gens, CPCS-I restarts or CHAPs are required when new CDIF exit routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-2. CDIF Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
Call Parameters • The codeline record format is specified by these parameters.	DKNCRCDP	DKNCDP
Compressed Codeline Record • This area must be referenced when the call parameters indicate ‘compressed codeline record’.	DKNCRDI	DIDSCT
Uncompressed Codeline Record • This area must be referenced when the call parameters indicate ‘uncompressed codeline record’.	DKNCRZD	ZDDSCT

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-3. CDIF Exit Routine Return Codes

Code (dec)	Request
+00	Write record to output data set.
+12	Terminate CDIF processing.
All others	Do not write record to output data set.

Restrictions

CDIF exit routines must be reentrant and reusable.

Example 1

Operation:

Use a CDIF routine to process compressed and uncompressed non-control records as follows:

1. Exclude records with a user code of C0 from being written to the output data set.
2. Terminate CDIF processing if an amount greater than $10,000.00 is detected.

User Exit Routine:

```
  ID DIVISION.
  PROGRAM-ID. CDIFEXT1.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  RETURN-CODES.
      05  RC-WRITE                    PIC 9(2) VALUE  0.
      05  RC-TERMINATE                PIC 9(2) VALUE 12.
      05  RC-EXCLUDE                  PIC 9(2) VALUE 20.

  01  AMOUNTS.
      05  AM-ZD-LIMIT                 PIC 9(5) VALUE 10000.
      05  AM-DI-LIMIT.
          10  FILLER                  PIC X(2) value X'0001'.
          10  FILLER                  PIC X(3) value X'000000'.

  01  DI-VALUES.
      05  DI-CONTROL                  PIC X(1) VALUE X'80'.

  01  USER-CODES.
      05  UC-EXCLUDE                  PIC X(1) VALUE X'C0'.
 /
  LINKAGE SECTION.
```

Figure 2-2 (Part 1 of 2). CDIF Exit Routine Example 1

```
 COPY DKNCATCB.

 COPY DKNCRCDP.

 COPY DKNCRDI                          REPLACING ==:DI:==
                                       BY        ==DI==.

 COPY DKNCRZD                          REPLACING ==:ZD:==
                                       BY        ==ZD==.
/
 PROCEDURE DIVISION                    USING APTCB
                                             CDIFP-PARMS
                                             ZC
                                             ZD.

     MOVE RC-WRITE                           TO RETURN-CODE.

     EVALUATE CDIFP-DEFTI-OUTPUT-FORMAT

         WHEN CDIFP-DEFTI-ZD
              IF  NOT ZD-CONTROL
                  IF  ZD-USER-FLAGS          =  UC-EXCLUDE
                      MOVE RC-EXCLUDE        TO RETURN-CODE
                  END-IF
                  IF  ZD-AMT                 >  AM-ZD-LIMIT
                      MOVE RC-TERMINATE      TO RETURN-CODE
                  END-IF
              END-IF

         WHEN CDIFP-DEFTI-ZC
              IF  DI-FLAG-2                  NOT = DI-CONTROL
                  IF  DI-USER-FLAG           =  UC-EXCLUDE
                      MOVE RC-EXCLUDE        TO RETURN-CODE
                  END-IF
                  IF  DI-AMT                 >  AM-DI-LIMIT
                      MOVE RC-TERMINATE      TO RETURN-CODE
                  END-IF
              END-IF

         WHEN OTHER
              MOVE RC-TERMINATE              TO RETURN-CODE

     END-EVALUATE.

     GOBACK.
```

Figure 2-2 (Part 2 of 2). CDIF Exit Routine Example 1

Example 2

Operation:

Use a CDIF routine to exclude non-control records with a user code of D0 from being written to the output data set. Processed records are in the compressed format.

User Exit Routine:

```
CDIFEXT2 CSECT
CDIFEXT2 AMODE 24
CDIFEXT2 RMODE 24
*
         SAVE  (14,12)             SAVE REGISTERS
         BASR  R12,0               BASE
         USING *,R12                 SUBROUTINE
*
         LA    R14,SAVEAREA        POINT TO OUR SAVE AREA
         ST    R14,8(R13)          STORE FORWARD AND
         ST    R13,4(R14)            BACKWARD SAVE AREA @
         LR    R13,R14             POINT TO NEW SAVE AREA
*
         L     R2,4(R1)            GET DKNCDIF CALL PARMS @
         USING CDIFDSCT,R2         MAP DKNCDIF CALL PARMS
         L     R3,8(R1)            GET COMPRESSED CODELINE RECORD @
         USING DIDSCT,R3           MAP COMPRESSED CODELINE RECORD
*
         LA    R15,0               SET 'WRITE RECORD' RETURN CODE
*
         CLI   DIFLAG2,DIMTCR      CONTROL CODELINE RECORD ?
         BE    RETURN              Y. EXIT PROGRAM
*
         CLI   DITYPEU,EXCLUDE     'EXCLUDE RECORD' USER CODE ?
         BNE   RETURN              N. EXIT PROGRAM
         LA    R15,8               SET 'EXCLUDE RECORD' RETURN CODE
*
RETURN   L     R13,SAVEAREA+4      RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)  RETURN TO DFTO
*
         DKNREGS                   REGISTER EQUATES
*
SAVEAREA DS    18F                 OUR SAVE AREA
*
EXCLUDE  EQU   X'D0'               'EXCLUDE FROM WRITING' USER CODE
*
CDIFDSCT DSECT                     DKNCDIF CALL PARMS
         COPY  DKNCDP
*
DIDSCT   DSECT                     COMPRESSED CODELINE RECORD
         COPY  DIDSCT
*
         END
```

Figure 2-3. CDIF Exit Routine Example 2

CSBU Exits 1/2 — Modify Edit Routine and Table Load Processing

These CSBU exits may be used to dynamically build the names of the edit routine and tables, and modify their load process. Exit 1 applies to 2-byte addressing edit routine and tables. Exit 2 applies to 4-byte addressing edit routine and tables.

Note: CSBU exits 1/2 routines have the same names as MICR exits 5/6 routines. Also, CSBU exits 1/2 and MICR exits 5/6 are activated simultaneously. See “MICR Exits 5/6 — Customize Document Processor’s Edit Routine and Table Load Processing (XF Sort Types)” on page 2-83 for information on activation, restrictions and linkage.

CYCL Exit — Validate Cycle and Endorse Dates

The CYCL exit may be used to perform additional cycle and endorse date validations.

CPCS-I edits cycle and endorse dates for a numeric year, a 01 through 12 month, and a valid day for the specified month, before calling the CYCL exit (if active).

Notes:

1. If active, the CYCL exit is called by both the CYCL task and the cycle control information update module (see “Accessing/Updating Cycle Control Information” on page 1-77).
2. Member DKNCYCLX of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding assembler CYCL exit routines.

Activation

The CYCL exit is a CPCS-I system-wide exit. The exit is active when a CYCL exit routine name is specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when a new CYCL exit routine version is installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-4. CYCL Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters	DKNCCYCX	CYCXDSCT

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs), and also in the call parameters:

Table 2-5. CYCL Exit Routine Return Codes

Code (dec)	Request
+00	Successful date validations.
All others	Unsuccessful date validations. • An error message, to be displayed on the CYCL screen, may be returned in this case.

The completion code may also be returned in the call parameters.

Example 1

Operation:

Verify that endorse dates specify the current date. It is assumed that the CPCS date format is YYYY/MM/DD. The message `ENDORSE DATE NOT TODAY'S DATE` is displayed and a return code of 4 is returned if the requirement is not met.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. CYCLEXT.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  ENDORSE-DATE.
     05  ED-YYYY                PIC  X(04).
     05                         PIC  X(01).
     05  ED-MM                  PIC  X(02).
     05                         PIC  X(01).
     05  ED-DD                  PIC  X(02).

 01  WORK-DATE.
     05  WD-YYYY                PIC  X(04).
     05  WD-MM                  PIC  X(02).
     05  WD-DD                  PIC  X(02).

 01  MESSAGE-1.
     05                         PIC  X(08) VALUE 'ENDORSE'.
     05                         PIC  X(05) VALUE 'DATE'.
     05                         PIC  X(04) VALUE 'NOT'.
     05                         PIC  X(09) VALUE 'TODAY''S'.
     05                         PIC  X(04) VALUE 'DATE'.

 01  RETURN-CODES.
     05  RC-GOOD-DATE           PIC  9(01) VALUE 0.
     05  RC-BAD-DATE            PIC  9(01) VALUE 4.
/
 LINKAGE SECTION.

 COPY DKNCCYCX.
/
 PROCEDURE DIVISION              USING DKNCYCX-PARMS.

     MOVE CYCX-ENDORSE-DATE      TO ENDORSE-DATE.
     MOVE ED-YYYY                TO WD-YYYY.
     MOVE ED-MM                  TO WD-MM.
     MOVE ED-DD                  TO WD-DD.
```

Figure 2-4 (Part 1 of 2). CYCL Exit Routine Example 1

```
      IF  WORK-DATE               NOT = FUNCTION CURRENT DATE (1:8)
          MOVE RC-BAD-DATE        TO CYCX-RETURN-CODE
          MOVE MESSAGE-1          TO CYCX-MESSAGE.
      ELSE
          MOVE RC-GOOD-DATE       TO CYCX-RETURN-CODE.

      GOBACK.
```

Figure 2-4 (Part 2 of 2). CYCL Exit Routine Example 1

Example 2

Operation:

Ensure that January 1 is never accepted as a cycle or endorse date in a US environment. The message `X11 DATE IS NEW YEARS DAY` is displayed on the CYCL screen if the requirement is not met.

Note: Member DKNCYCLX of CPCSI.V01R01.SDKNSAM2 contains the assembler code for this example.

DFTI Exit 1 — Validate DEFT Input Initialization

The DFTI exit 1 may be used to perform additional DFTI initialization validations, and stop DFTI processing if necessary.

Message DFTI 30024, which includes the exit routine name and the exit routine return code, is displayed on the CPCS-I supervisor's terminal and written to the scroll log when DFTI is ended by the exit 1 routine (see *CPCS-I Messages and Codes* for more information).

Note: Member DKNDFTU1 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding assembler DFTI exit 1 routines.

Activation

A different DFTI 1 exit routine may be specified for each financial institution. The exit is active when one or more DFTI exit routine names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

The DFTI exit 1 also is activated when the name of a DFTI exit 1 routine is specified in the DEFT profile (see the chapter, "System and Application Profiles," in the *CPCS-I Customization Guide* for a description of the DEFT profile parameters). The DEFT profile specifications take precedence over the bank control file specifications.

The WORK parameter in the DKNBLDL entry for DKNDFTI defines the amount of user work storage allocated for the DFTI exit 1 routine. The same user work area is used by the DFTI exit 2.

No re-gens, CPCS-I restarts or CHAPs are required when new DFTI exit 1 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-6. DFTI Exit 1 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
User Work Area		
DEFT Profile Parameters	DKNCRTPF	DKNDFTPF
Input Data Set Header Record	DKNCRTHT	DKNDFTHT

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-7. DFTI Exit 1 Routine Return Codes

Code (dec)	Request
+00	Resume DFTI processing.
All others	Terminate DFTI processing.

Restrictions

DFTI exit 1 routines must be reentrant and reusable.

Example 1

Operation:

Use the DFTI initialization exit to stop DFTI processing with a return code of 12 if the DEFT profile bank number is smaller than 005 or greater than 010, or if the input data set header record indicates ‘electronic data only’.

Assembler User Exit Routine:

```
DFTIEX11 CSECT
DFTIEX11 AMODE 31
DFTIEX11 RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE ROUTINE
*
         LM    R2,R4,0(R1)            GET CONTROL BLOCK @'S
         USING WRKA,R2                MAP USER WORK AREA
         USING PROFDSCT,R3            MAP DEFT PROFILE BLOCK
         USING DHDREC,R4              MAP HEADER RECORD
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         CLC   DPF01_BANKNUM,BANK1ST  VALID PROFILE BANK NUMBER ?
         BL    STOPDFTI               N. REQUEST DFTI STOP
         CLC   DPF01_BANKNUM,BANKLST  VALID PROFILE BANK NUMBER ?
         BH    STOPDFTI               N. REQUEST DFTI STOP
         CLC   DHDEONLY,EDONLY        ELECTR-DATA-ONLY ON IN HDR ?
         BE    STOPDFTI               Y. REQUEST DFTI STOP
         XR    R15,R15                RESUME DFTI PROCESSING
         B     RETURN                 EXIT PROGRAM
*
STOPDFTI LA    R15,12                 END DFTI WITH RC=12
*
```

Figure 2-5 (Part 1 of 2). Assembler DFTI Exit 1 Routine Example 1

```
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN TO DFTI
*
         DKNREGS                       REGISTER EQUATES
*
BANK1ST  DC    C'005'                  BANK NUMBER LOWER LIMIT
BANKLST  DC    C'010'                  BANK NUMBER UPPER LIMIT
EDONLY   DC    C'Y'                    ELECTRONIC DATA ONLY
*
WRKA     DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       OUR SAVE AREA
*
PROFDSCT DSECT                         DEFT PROFILE BLOCK
         COPY  DKNDFTPF
*
HDRDSCT  DSECT                         DEFT HEADER RECORD
         COPY  DKNDFTHT
*
         END
```

Figure 2-5 (Part 2 of 2). Assembler DFTI Exit 1 Routine Example 1

COBOL User Exit Routine:

```
  ID DIVISION.
  PROGRAM-ID. DFTIEX11.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  BANK-NUMBERS.
      05  BN-LOWER-LIMIT               PIC X(03) VALUE '005'.
      05  BN-UPPER-LIMIT               PIC X(03) VALUE '010'.

  01  RETURN-CODES.
      05  RC-STOP                      PIC 9(02) VALUE 12.
 /
  LINKAGE SECTION.

  01  USER-WORK-AREA                   PIC X(90).

  COPY DKNCRTPF.

  COPY DKNCRTHT.
 /
  PROCEDURE DIVISION                    USING USER-WORK-AREA
                                              DEFT-PROFILE-PARMS
                                              DEFT-HEADER.
```

Figure 2-6 (Part 1 of 2). COBOL DFTI Exit 1 Routine Example 1

```
      IF  DPP01-BANK-NUMBER            <  BN-LOWER-LIMIT
       OR DPP01-BANK-NUMBER            >  BN-UPPER-LIMIT
       OR ELECTRONIC-DATA-ONLY
          MOVE RC-STOP                 TO RETURN-CODE.

      GOBACK.
```

Figure 2-6 (Part 2 of 2). COBOL DFTI Exit 1 Routine Example 1

DFTI Exit 2 — Modify DEFT Input Processing

The DFTI exit 2 allows you to:

- Inspect/modify each codeline record before it is written to the output string.
- Exclude codeline records from being written to the output string.
- Stop DFTI processing.

The exit routine is called, for each item processed, immediately following return from the call to the host user edit routine (if the user editing option is on in the DEFT profile).

Messages DFTI 30024, which includes the exit routine name and the exit routine return code, and DFTI 30010, are displayed on the CPCS-I supervisor's terminal and written to the scroll log when DFTI is ended by the exit 2 routine (see DFTI messages in *CPCS-I Messages and Codes* for more information).

Note: Member DKNDFTU2 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding assembler DFTI exit 2 routines.

Activation

A different DFTI 2 exit routine may be specified for each financial institution. The exit routine is active when one or more DFTI exit 2 routine names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

The DFTI exit 2 also is active when the name of an exit 2 routine is specified in the DEFT profile (see the chapter, "System and Application Profiles," in the *CPCS-I Customization Guide* for a description of the DEFT profile parameters). The DEFT profile specifications take precedence over the bank control file specifications.

The WORK parameter in the DKNBLDL entry for DKNDFTI defines the amount of user work storage allocated for the DFTI exit 2 routine. The same user work area is used by the DFTI exit 1.

No re-gens, CPCS-I restarts or CHAPs are required when new DFTI exit 2 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-8. DFTI Exit 2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
User Work Area		
DEFT Profile Parameters	DKNCRTPF	DKNDFTPF
Input Data Set Header Record	DKNCRTHT	DKNDFTHT
Compressed Codeline Record	DKNCRDI	DIDSCT

The exit routine may return the following completion codes in register 15 (assembler programs) or in the RETURN-CODE special register (COBOL programs):

Table 2-9. DFTI Exit 2 Routine Return Codes

Code (dec)	Request
+00	Write record to output string.
+04	Do not write record to output string.
> +04	Terminate DFTI processing.

Restrictions

DFTI exit 2 routines must be reentrant and reusable.

Example 1

Operation:

Use the DFTI exit 2 to:

1. Stop DFTI processing if a routing/transit number other than 32280001 is received from sending bank 001.
2. Assign a user code of E9 to detail records having a user code in the range E0–E7.
3. Exclude records with a user code of B2 from being written to the output string.

Assembler User Exit Routine:

```
DFTIEX21 CSECT
DFTIEX21 AMODE 31
DFTIEX21 RMODE 24
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE ROUTINE
*
         L     R2,0(R1)             GET USER WORK AREA @
         USING WRA,R2               MAP USER WORK AREA
         L     R3,8(R1)             GET HEADER RECORD @
         USING DHDREC,R3            MAP HEADER RECORD
         L     R4,12(R1)            GET COMPRESSED RECORD @
         USING DIDSCT,R4            MAP CODELINE RECORD
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
```

Figure 2-7 (Part 1 of 3). Assembler DFTI Exit 2 Routine Example 1

```
         CLI   DIFLAG2,DIMTCR        CONTROL RECORD ?
         BE    RESDFTI               Y. WRITE TO OUTPUT STRING
         CLC   DHDPBNK#,BANK1        BANK 001 ?
         BNE   TESTRNUC              N. SEE IF USER CODE IS IN RANGE
         CLC   DIABA,RT1             VALID R/T NUMBER ?
         BE    TESTRNUC              Y. SEE IF USER CODE IS IN RANGE
         LA    R15,8                 END DFTI PROCESSING
         MVI   FREESW,FREEWRK        PRIVATE WORK AREA MUST BE FREED
         B     RETURN                EXIT PROGRAM
*
TESTRNUC CLC   DITYPEU,LOWTYPEU      USER CODE < USER CODE RANGE ?
         BL    TESTEXUC              Y. SEE IF IT MUST BE WRITTEN
         CLC   DITYPEU,HIGTYPEU        USER CODE > USER CODE RANGE ?
         BH    RESDFTI               Y. WRITE TO OUTPUT STRING
         MVC   DITYPEU,NEWTYPEU      SET NEW USER CODE
         B     RESDFTI               WRITE TO OUTPUT STRING
*
TESTEXUC CLC   DITYPEU,EXCTYPEU      EXCLUDE USER CODE TYPE ?
         BNE   RESDFTI               N. WRITE TO OUTPUT STRING
         LA    R15,4                 DO NOT WRITE TO OUTPUT STRING
         B     RETURN                EXIT PROGRAM
*
RESDFTI  XR    R15,R15               RESUME DFTI PROCESS
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN TO DFTI
*
         DKNREGS                     REGISTER EQUATES
*
BANK1    DC    C'001'                BANK 001 CODE
RT1      DC    X'32280001'           BANK 001 VALID R/T NUMBER
LOWTYPEU DC    X'E0'                 USER CODE LOWER LIMIT
HIGTYPEU DC    X'E7'                 USER CODE UPPER LIMIT
NEWTYPEU DC    X'E9'                 NEW USER CODE
EXCTYPEU DC    X'B2'                 EXCLUDE USER CODE
*
WRA      DSECT                       USER WORK AREA
SAVEAREA DS  18F                     . OUR SAVE AREA
*
HDRDSCT  DSECT                       DEFT HEADER RECORD
         COPY  DKNDFTHT
*
DIDSCT   DSECT                       COMPRESSED CODELINE RECORD
         COPY  DIDSCT
*
         END
```

Figure 2-7 (Part 2 of 3). Assembler DFTI Exit 2 Routine Example 1

COBOL User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. DFTIEX21.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  BANK-ID-1                        PIC X(03) VALUE '001'.
 01  RT-1                             PIC X(04) VALUE X'32280001'.
 01  CONTROL-RECORD                   PIC X(01) VALUE X'80'.

 01  USER-CODES.
     05  UC-EXCLUDE                   PIC X(01) VALUE X'B2'.
     05  UC-LOWER-LIMIT               PIC X(01) VALUE X'E0'.
     05  UC-UPPER-LIMIT               PIC X(01) VALUE X'E7'.
     05  UC-NEW                       PIC X(01) VALUE X'E9'.

 01  RETURN-CODES.
     05  RC-EXCLUDE                   PIC 9(01) VALUE 4.
     05  RC-STOP                      PIC 9(01) VALUE 8.
/
 LINKAGE SECTION.

 01  USER-WORK-AREA                   PIC X(90).

 COPY DKNCRTPF.

 COPY DKNCRTHT.

 COPY DKNCRDI                         REPLACING ==:DI:==
                                      BY        ==DI==.
/
 PROCEDURE DIVISION                   USING USER-WORK-AREA
                                            DEFT-PROFILE-PARMS
                                            DEFT-HEADER
                                            DI.

     IF  DI-FLAG-2                    NOT = CONTROL-RECORD

         IF  DH-BANK-NUMBER           = BANK-ID-1
             IF  DI-ROUTING-TRANSIT   NOT = RT-1
                 MOVE RC-STOP         TO RETURN-CODE
             END-IF
         END-IF

         IF  RETURN-CODE              NOT = RC-STOP
             IF  DI-USER-FLAG         = UC-EXCLUDE
                 MOVE RC-EXCLUDE      TO RETURN-CODE
             ELSE
                 IF  DI-USER-FLAG     >= UC-LOWER-LIMIT
                     IF  DI-USER-FLAG <= UC-UPPER-LIMIT
                         MOVE UC-NEW  TO DI-USER-FLAG.

     GOBACK.
```

Figure 2-8. COBOL DFTI Exit 2 Routine Example 1

DFTO Exit — Modify DEFT Output Processing

The DFTO user exit routine allows you to:

- Insert a record into the DFTO output data set, and receive again the detail record that was sent previously.
- Exclude records from being written to the DFTO output data set.
- Inspect/modify each record before it is written to the DFTO output data set.
- Abnormally end DFTO processing.

DFTO calls the exit routine, if active, before writing each compressed codeline record to the output transmission data set.

DFTO notifies the exit routine of its first and last call, the first record for a new string and the first record for a new bundle. The call-parameters include information associated with the current endpoint being processed, such as the sending bank number, the capture cycle, the cycle date, the name of the string currently being processed, and the name of the requested endpoint. This information may be used to build trailer and header records to be inserted in the output transmission data set.

The first record being processed is passed on the first exit call. The last record being processed is passed again on the last exit call.

Message DFTOX30002 is displayed on the CPCS-I supervisor's terminal and written to the scroll log (see DFTO messages in *CPCS-I Messages and Codes* for more information), the DFTO task abends with an 0C4 MVS/ESA system code, and the transmission data set is not created, when DFTO is ended by the exit routine.

Note: Member DEFTOEXT of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding assembler DFTO user exit routines.

Activation

A different DFTO exit routine may be specified for each financial institution. The exit is active when one or more DFTO exit routine names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when new DFTO exit routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-10. DFTO Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters • These parameters include a fullword aligned user work area. • DFTO processing modification requests are made via the status field in these parameters.	DKNDFTOC	DKNDFTOA
Compressed Codeline Record	DKNCRDI	DIDSCT
Application Task Control Block	DKNCATCB	DKNAPTCB

Restrictions

The DFTO exit routine must be reentrant.

Example 1

Operation:

Use the DFTO exit to:

1. Build and insert a trailer record, containing the bundle number, the bundle item count and the bundle total amount, at the end of each bundle in the DFTO output transmission data set.
2. Exclude records with a user code of E0 from being written to the DFTO output transmission data set.

User Exit Routine:

```
  ID DIVISION.
  PROGRAM-ID. DEFTOEXT.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  USER-CODES.
      05  UC-EXCLUDE                     PIC X(1) VALUE X'E0'.

  01  SWITCHES.
      05  SW-RESENT-PROCESSING           PIC X(1) VALUE 'Y'.
          88  SW-PROCESS-RECORD                   VALUE 'Y'.
          88  SW-BYPASS-RECORD                    VALUE 'N'.

  01  DI-VALUES.
      05  DI-CONTROL                     PIC X(1) VALUE X'80'.
```

Figure 2-9 (Part 1 of 3). DFTO Exit Routine Example 1

```
 01  BUNDLE-TRAILER.
     05  FILLER                          PIC X(6) VALUE 'BUNDLE'.
     05  BT-BUNDLE-NBR                   PIC 9(5) VALUE 0 COMP-3.
     05  BT-ITEM-COUNT                   PIC 9(7)         COMP-3.
     05  BT-TOTAL-AMT                    PIC 9(9)         COMP-3.

 01  WORK-FIELDS.

     05  WF-AMT.
         10  WF-AMT-1                    PIC X(5).
         10  WF-AMT-2                    PIC 9(1)         COMP-3.
     05  WF-AMT-9                        REDEFINES        WF-AMT
                                         PIC 9(11)        COMP-3.

     05  WF-ITEM-COUNT                   PIC 9(7) VALUE 0 COMP-3.

     05  WF-TOTAL-AMT                    PIC 9(9) VALUE 0 COMP-3.

 COPY DKNCRDI                            REPLACING ==:DI:==
                                         BY        ==DI==.
/
 LINKAGE SECTION.

 COPY DKNCATCB.
/
 PROCEDURE DIVISION                   USING   DFTO-USER-EXIT-PARMS
                                              DFTO-DI-IMAGE
                                              APTCB.

     MOVE DFTO-DI-IMAGE                       TO DI.
     SET  DFTO-WRITE-RECORD                   TO TRUE.

     IF  DFTO-LAST-TIME-CALL
         PERFORM 2000-CREATE-TRAILER
     ELSE
         IF  SW-PROCESS-RECORD
             IF  DFTO-FIRST-RECORD-NEW-BUNDLE
                 PERFORM 2000-CREATE-TRAILER
             END-IF
             IF  DI-USER-FLAG                 =  UC-EXCLUDE
                 SET DFTO-RECORD-TO-BE-DELETED TO TRUE
             ELSE
                 IF  DI-FLAG-2                NOT =  DI-CONTROL
                     PERFORM 1000-UPDATE-COUNTERS
                 END-IF
             END-IF
         ELSE
             SET SW-PROCESS-RECORD            TO TRUE.

     GOBACK.

 1000-UPDATE-COUNTERS.
```

Figure 2-9 (Part 2 of 3). DFTO Exit Routine Example 1

```
     ADD  1                                  TO WF-ITEM-COUNT.
     MOVE 0                                  TO WF-AMT-2.
     MOVE DI-AMT                             TO WF-AMT-1.
     COMPUTE WF-AMT-9                        =  WF-AMT-9
                                             /  10.
     ADD  WF-AMT-9                           TO WF-TOTAL-AMT.

 2000-CREATE-TRAILER.

     ADD  1                                  TO BT-BUNDLE-NBR.
     MOVE WF-ITEM-COUNT                      TO BT-ITEM-COUNT.
     MOVE WF-TOTAL-AMT                       TO BT-TOTAL-AMT.
     MOVE BUNDLE-TRAILER                     TO DFTO-DI-IMAGE.
     SET  DFTO-INSERTING-RESEND-DI-IMAGE
          SW-BYPASS-RECORD                   TO TRUE.
     MOVE 0                                  TO WF-ITEM-COUNT
                                                WF-TOTAL-AMT.
```

Figure 2-9 (Part 3 of 3). DFTO Exit Routine Example 1

Example 2

Operation:

Use the DFTO exit to:

Build and insert an output transmission data set header record, that contains the sending bank number, the capture cycle number and date, and the endpoint number.

1. Assign a user code of E3 to detail records having an amount greater than $5,000.00.
2. Stop DFTO processing if a user code greater than F0 is detected.

User Exit Routine:

```
DEFTOEXT CSECT
DEFTOEXT AMODE 31
DEFTOEXT RMODE ANY
*
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 BASE ROUTINE
*
         L     R6,0(R1)              GET DFTO CALL-PARMS @
         USING DFTO_PDSCT,R6         MAP DFTO CALL-PARMS
         L     R2,4(R1)              GET COMPRESSED CODELINE REC @
         USING DIDSCT,R2             MAP COMPRESSED CODELINE REC
         L     R8,8(R1)              GET APTCB @
         USING APTCB,R8              MAP APTCB
         LA    R3,DFTO_USRWORK       GET WORK AREA @
         USING WRA,R3                MAP WORK AREA
*
```

Figure 2-10 (Part 1 of 3). DFTO Exit Routine Example 2

```
         CLI   STORSW,STORDONE        WORK AREA ALREADY ALLOCATED ?
         BE    CHSAREAS               Y. SKIP 'STORAGE OBTAIN'
         STORAGE OBTAIN,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         LTR   R15,R15                SUCCESSFUL AREA ALLOCATION ?
         BZ    SETWRKSW               Y. CONTINUE
         MVI   DFTO_RSTAT,DFTO_ABORT  SET STOP-DFTO REQUEST CODE
         B     RETURN                 EXIT PROGRAM
*
SETWRKSW MVI   STORSW,STORDONE        SET WORK AREA ALLOCATED FLAG
         USING WRK,R5                 MAP PRIVATE WORK AREA
*
CHSAREAS LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         CLI   DFTO_PSTAT,DFTO_LAST   LAST CALL ?
         BNE   TSTFIRST               N. SEE IF FIRST CALL
         MVI   FREESW,FREEWRK         SET FREE WORK AREA INDICATOR
         B     RETURN                 EXIT PROGRAM
*
TSTFIRST CLI   DFTO_PSTAT,DFTO_FIRST  FIRST CALL ?
         BNE   TESTUC                 N. PROCESS RECORD
         MVC   HDRID,HDRNAME          BUILD HEADER REC - ID
         MVC   HDRBANK#,DFTO_FRMBNK                    - BANK #
         MVC   HDRCYL#,DFTO_CYCLNUM                    - CYCLE #
         MVC   HDRCYLDT,DFTO_CYCDATE                   - CYCLE DATE
         MVC   HDREPT,DFTO_ENPTNUM                     - ENDPOINT #
         XC    DI,DI                  CLEAR CODELINE RECORD AREA
         MVC   DI(L'HDRREC),HDRREC    HEADER RECORD TO CODELINE RECRD
         MVI   DFTO_RSTAT,DFTO_RECINS SET INSERT-RESEND REQUEST CODE
         B     RETURN                 EXIT PROGRAM
*
TESTUC   MVI   DFTO_RSTAT,DFTO_NORMAL SET DEFAULT WRITE REQUEST CODE
         CLI   DIFLAG2,DIMTCR         CONTROL RECORD ?
         BE    RETURN                 Y. EXIT PROGRAM
         CLC   DITYPEU,UCLIMIT        USER CODE INVALID ?
         BNH   TESTAMT                N. PROCESS RECORD
         MVI   DFTO_RSTAT,DFTO_ABORT  SET STOP-DFTO REQUEST CODE
         MVI   FREESW,FREEWRK         SET FREE WORK AREA INDICATOR
         B     RETURN                 EXIT PROGRAM
*
TESTAMT  CLC   DIAMT,HAMT             HIGH AMOUT ?
         BNH   RETURN                 N. EXIT PROGRAM
         MVC   DITYPEU,HAMTUC         SET HIGH AMOUNT USER CODE
*
RETURN   L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         CLI   FREESW,FREEWRK         MUST WORK AREA BE FREED ?
         BNE   EXITPR                 N. EXIT PROGRAM
         STORAGE RELEASE,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         XC    STORSW,STORSW          RESET WRKAREA ALLOCATED SWITCH
         XC    FRESW,FREESW           RESET FREE WORK AREA SWITCH
*
EXITPR   XR    R15,R15                CLEAR RETURN CODE
         XRETURN (14,12),,RC=(15)     RETURN TO DFTO
*
```

Figure 2-10 (Part 2 of 3). DFTO Exit Routine Example 2

```
         DKNREGS                       REGISTER EQUATES
*
HDRNAME  DC    C'HDR'                  HEADER RECORD ID
HAMT     DC    X'0000500000'           HIGH AMOUNT VALUE
HAMTUC   DC    X'E3'                   HIGH AMOUNT USER CODE
UCLIMIT  DC    X'F0'                   USER CODE LIMIT
*
WRK      DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       . OUR SAVE AREA
WRKSIZE  EQU   *-WRK                   LENGTH OF PRIVATE WORK AREA
*
WRA      DSECT                         USER WORK AREA
STORSW   DS    X                       . PRIVATE WRKAREA ALLOC SWITCH
STORDONE EQU   X'01'                     - PRIVATE WORKAREA ALLOCATED
FREESW   DS    X                       . FREE PRIVATE WORKAREA SWITCH
FREEWRK  EQU   X'01'                     - FREE PRIVATE WORK AREA
*
HDRREC   DS   0CL22                    . HEADER RECORD
HDRID    DS    CL3                       - HEADER ID
HDRBANK# DS    CL3                       - SENDING BANK NUMBER
HDRCYL#  DS    CL2                       - PROCESSING CYCLE NUMBER
HDRCYLDT DS    CL6                       - PROCESSING CYCLE DATE
HDREPT   DS    CL8                       - DFTO ENDPOINT NUMBER
*
         COPY  DKNDFTOA                DFTO CALL-PARMS
         COPY  DKNAPTCB                APTCB
*
DIDSCT   DSECT                         COMPRESSED CODELINE RECORD
         COPY  DIDSCT
*
         END
```

Figure 2-10 (Part 3 of 3). DFTO Exit Routine Example 2

Example 3

Operation:

Use the DFTO exit to insert an output transmission data set header record that has the format required by the DFTI task.

User Exit Routine:

Note: Member DFTOEXT of CPCSI.V01R01.SDKNSAM2 contains the assembler code for this example.

DFTP Exit — Analyze DFTP Activity

The DFTP exit may be used to review the results of a DFTP run and to optionally display a message.

Counters are passed to the DFTP exit that summarize DFTP activity. These counters show:

- How many eligible DEFT input control file records were submitted to DFTI
- How many completed records (records that had been successfully processed by DFTI) were deleted
- How many queued records (records that had been submitted to DFTI but which for some reason, possibly due to a system crash, had never been processed) were resubmitted to DFTI.

Any messages to be generated by the DFTP exit must have been previously defined on the System Messages data set. For more information, see:

- "DKNSMSG–System Message Handler" in the *CPCS-I Customization Guide*
- "DKNSMSG–System Message Handler" in the *CPCS-I Programming Guide*
- "Issuing Messages" on page 1-39

All the DKNSMSG facilities for providing override fields, and for routing messages to multiple destinations, exist within this exit.

Note: Member DKNDFPU1 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding assembler DFTP exit routines.

Activation

The DFTP exit is activated when the name of a DFTP exit routine is specified in the DFTP profile. (See "System and Application Profiles" in the *CPCS-I Customization Guide* for a description of the DFTP profile parameters.)

No regenerations, CPCS-I restarts, or CHAPs are required when the new DFTP exit routine versions are installed.

Linkage

The standard linkage conventions are followed.

The following interface control blocks are used to invoke DKNDFTP:

Table 2-11. DKNDFTP Exit Routine Parameter List

Size	Content
Fullword	DKNDFTP Call Parameters
Fullword	DFTP Record

The Counter Array is defined as follows:

```
COUNTERS DSECT          COUNT of record types processed:
ELIGIBLE DS    H        ... ELIGIBLE RECORDS
COMPLETE DS    H        ... COMPLETE RECORDS
QUEUED   DS    H        ... QUEUED RECORDS
```

The Start Switches are defined as follows:

```
STARTSWS DSECT           DFTP START SWITCH ---
DFTPSTRT DS    X           HOW DKNDFTP WAS STARTED:
OPERATOR EQU   B'00000001' ... MANUALLY FROM TERMINAL
ESMECYC  EQU   B'00000010' ... BY ESM DURING END CYCLE
ESMTIMER EQU   B'00000100' ... BY ESM START TIMER
INITIAL  EQU   B'00001000' ... 1ST RUN SINCE CPCS IPL
ESMCOLD  EQU   B'00010000' ... BY ESM DURING COLD START
ESMANUAL EQU   B'00100000' ... MANUAL START VIA ESM
```

The SMSG parameter block is defined as follows:

```
SMSGPARM DSECT           OPTIONAL MESSAGE:
DESTCODE DS    AL2       ... DEST CODE(S) (NOTE: MAY
*                            BE OR'D TOGETHER)
SMPDRIN  EQU   X'0001'       ... RETURN MESSAGE TO DFTP
SMPDSUPV EQU   X'0002'       ... SEND MESSAGE TO SUPV
SMPDSCRL EQU   X'0004'       ... SEND MESSAGE TO SCROLL
SMPDWTO  EQU   X'0008'       ... WTO DISPLAY
SMPDMICR EQU   X'0010'       ... SEND MESSAGE TO MICR
SMPDINSC EQU   X'0020'       ... SEND MESSAGE TO INSC
*
PROGNAME DS    CL8... SUBSYSTEM / PROGRAM NAME
MSGNUMBR DS    CL5       ... MESSAGE NUMBER
INSERTBF DS    CL120     ... INSERT BUFFER
```

The exit routine may return the following completion codes in register 15:

Table 2-12. DFTP Exit Routine Return Codes

Code (dec)	Request
+00	End DFTP without taking any special action.
+04	Display a message before ending DFTP. Before returning, the DFTP exit should have loaded PROGNAME and MSGNUMBR with the DKNSMSG ID of the message, and should have OR'ed into DESTCODE a bit pattern representing where the message should be sent. If the message has override fields, the modifications should have been loaded into INSERTBF. Each INSERTBF field should be terminated with a X'00'; each should be concatenated, one after the other. An extra X'00' should follow the X'00' of the last field to indicate the end of the buffer.

Restrictions

The DFTP exit routines must be reentrant and reusable.

Example 1

Operation

Check to see that at least one eligible record had been submitted to DFTP. If none were, display a message.

Assembler User Exit Routine

```
DKNDFPU1 CSECT
DKNDFPU1 AMODE 31
DKNDFPU1 RMODE 24
*
         SAVE  (14,12)          SAVE REGISTERS
         BASR  R12,0            GET ROUTINE @
         USING *,R12            BASE ROUTINE
*
         LM    R8,R10,0(R1)     GET CONTROL BLOCK @'S
         USING COUNTERS,R8      MAP COUNTER ARRAY
         USING STARTSW,R9       MAP START SWITCHES
         USING SMSGPARM,R10     MAP SMSG PARM BLOCK
*
         LH    R2,ELIGIBLE      FETCH ELIGIBLE CTR.
         LTR   R2,R2            NON-ZERO ?
         BNZ   B99000           Y. EXIT W/O MESSAGE
 *
         MVC   PROGNAME,=CL8'   DFPU1'    MESSAGE
         MVC   MSGNUMBR,=CL5'30099'       DFPU130099
         MVC   DESTCODE,=AL2(SMPDRTN+SMPDSUPV)
         B     B99004           GO DISPLAY MESSAGE
*
B99000   DS    0H
         SLR   R15,R15          ZERO RETURN CODE
         B     B99999           GO TO RETURN
*
B99004   DS    0H
         LA    R15,4            SET RETURN CODE
*
B99999   DS    0H
         XRETURN (14,12),,RC=(15)   END OF DFPU1
*
         LTORG
*
COUNTERS DSECT                  RECORD TYPES:
ELIGIBLE DS    H                ... ELIGIBLE RECORDS
QUEUED   DS    H                ... QUEUED RECORDS
*
```

Figure 2-11 (Part 1 of 2). DFTP Exit Routine Example 1

```
STARTSW  DSECT                 DFTP START SWITCH --
DFTPSTRT DS    X                 HOW DKNDFTP STARTED
OPERATOR EQU   B'00000001'       ... MANUAL START
ESMECYC  EQU   B'00000010'       ... ESM END CYCLE
ESMTIMER EQU   B'00000100'       ... ESM START TIMER
INITIAL  EQU   B'00001000'       ... AFTER IPL
ESMCOLD  EQU   B'00010000'       ... ESM COLD START
ESMANUAL EQU   B'00100000'       ... ESM MANUAL
*
SMSGPARM DSECT                 MESSAGE BLOCK
DESTCODE DS    AL2             ... DESTINATION CODE
SMPDRTN  EQU   X'0001'              ... DKNDFTP
SMPDSUPV EQU   X'0002'              ... SUPV TUBE
SMPDSCRL EQU   X'0004'              ... SCROLL LOG
SMPDWTO  EQU   X'0008'              ... WTO
SMPDMICR EQU   X'0010'              ... MICR SUPV
SMPDINSC EQU   X'0020'              ... INSC SUPV
PROGNAME DS    CL8             ... PROGRAM NAME
MSGNUMBR DS    CL5             ... MESSAGE NUMBER
INSERTBF DS    CL120           ... INSERT BUFFER
*
         END
```

Figure 2-11 (Part 2 of 2). DFTP Exit Routine Example 1

FSCN Exit — Modify FSCN Processing

The FSCN user exit allows to:

- Inspect/modify each record before it is written to the FSCN output data set.
- Exclude selected records from being written to the FSCN output data set.
- Stop further FSCN processing.

The exit routine can perform first or last call processing based on the value of a passed call-sequence indicator.

Note: Member FSCNUEX1 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding MICR user exit routines in assembler language.

Inspecting/Modifying Records

The passed FSCN record contains the compressed codeline record.

Any field, including the default sort key (X'00F1F2F3') and the default user area (blanks), can be modified.

Excluding Records

A record can be bypassed by changing the call-status parameter (see Linkage below) to ‘SK’ (SKIP).

Stopping Processing

Processing can be stopped by passing a return code other than zero. When this occurs, DKNFSCN ends with a return code of 12.

A provided 30-byte message area can be used to describe the reason why processing is being ended. You have the option of placing a message number (which must be defined in the system message data set) in the first 5 bytes of the message area, instead of the actual message. In either case, the message is displayed on the supervisor’s terminal and written to the scroll log. Message FSGFSCN 30006 is always generated before the exit routine message. If a message text, rather than a message number, is returned, it is included as part of message FSGFSCN 30007 (see *CPCS-I Messages and Codes* for more information).

Activation

A different FSCN exit routine may be specified for each financial institution. The exit is active when one or more FSCN exit routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when new FSCN exit routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-13. FSCN Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters	DKNCFSCP	DKNFSCP
FSCN Record	DKNCRFSC	DKNFSC

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-14. FSCN Exit Routine Return Codes

Code (dec)	Request
+00	Resume FSCN processing.
All others	Terminate FSCN processing.

Example 1

Operation:

Use the FSCN exit to:

1. Stop processing if a record has a process code greater than 0000999999, returning the text “INVALID PC; SEQUENCE = xxxxxx”, where xxxxxx is the FSCN record sequence number, for its inclusion in the FSCN stop message.
2. Exclude records with a user code of 6B from being written to the output data set.
3. Set a sort key of zeros for records having a user code of 43.

User Exit Routine:

```
  ID DIVISION.
  PROGRAM-ID. FSCNUEX1.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  USER-CODES.
      05  UC-BYPASS                    PIC X(1) VALUE X'6B'.
      05  UC-NOSORT                    PIC X(1) VALUE X'43'.
```

Figure 2-12 (Part 1 of 2). FSCN Exit Routine Example 1

```
 01  PROCESS-CODES.
     05  PC-LIMIT                     PIC X(3) VALUE X'AA8888'.

 01  FSCN-REQUESTS.
     05  FR-SKIP                      PIC X(2) VALUE 'SK'.

 01  SORT-KEYS.
     05  SK-NULL                      PIC 9(8) VALUE 0 COMP.

 01 RETURN-CODES.
     05  RC-GOOD                      PIC 9(1) VALUE 0.
     05  RC-STOP                      PIC 9(1) VALUE 8.

 01  STOP-MESSAGE.
     05  FILLER                       PIC X(8) VALUE 'INVALID'.
     05  FILLER                       PIC X(4) VALUE 'PC;'.
     05  FILLER                       PIC X(9) VALUE 'SEQUENCE'.
     05  FILLER                       PIC X(2) VALUE '='.
     05  SM-SEQUENCE                  PIC 9(6).

 COPY DKNCRDI                          REPLACING ==:DI:==
                                       BY        ==DI==.
/
 LINKAGE SECTION.

 COPY DKNCFSCP.

 COPY DKNCRFSC.
/
 PROCEDURE DIVISION                    USING FSCN-CALL-PARMS
                                             FSCN-RECORD.

     IF  NOT FSCN-HDR-01
         IF  NOT FSCN-HDR-02
             MOVE FSCN-ITEM-MDS-IMAGE      TO DI
             IF  DI-PROCESS-CONTROL        >  PC-LIMIT
                 MOVE FSCN-I-SEQ-NBR       TO SM-SEQUENCE
                 MOVE STOP-MESSAGE         TO FSCN-CALL-MSG
             ELSE
                 IF  DI-USER-FLAG          =  UC-BYPASS
                     MOVE FR-SKIP          TO FSCN-CALL-STATUS
                     MOVE RC-GOOD          TO RETURN-CODE
                 ELSE
                     IF  DI-USER-FLAG      =  UC-NOSORT
                         MOVE SK-NULL      TO FSCN-SORT-KEY
                         MOVE RC-GOOD      TO RETURN-CODE.

     GOBACK.
```

Figure 2-12 (Part 2 of 2). FSCN Exit Routine Example 1

Example 2

Operation:

Create an FSCN exit routine that excludes control records from being written to the FSCN output data set.

User Exit Routine:

Note: Member FSCNUEX1 of CPCSI.V01R01.SDKNSAM2 contains the assembler code for this example.

HCDM Exit 1 — Modify HCDM Profile Parameters

This user exit allows the modification of the HCDM profile parameters, at run time, before host codeline data matching.

Note: Member HCMX001 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding COBOL HCDM user exit 1 routines.

Activation

A different HCDM exit 1 routine may be specified for each financial institution. The exit is active when its name is specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide*). For more information concerning the HCDM record profiles and record names, see the chapter, “System and Application Profiles,” in the *CPCS-I Customization Guide*.

No re-gens, CPCS-I restarts or CHAPs are required when new HCDM exit 1 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-15. HCDM Exit 1 Communication Control Block

Control Block	COBOL Copybook	Assembler Macro/ Copybook
HCDM Profile Parameters	DKNHCMC0	DKNHCMA0

Restrictions

The HCDM exit 1 routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the HCDM exit 1 to override the first digit of the reconciliation string name with an ‘8’.

User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. HCMX001.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  SUB                          PIC 9(2)             COMP.
 01  STRING-NAME-SIZE             PIC 9(2)  VALUE  17.
 01  RECONCILIATION-ID            PIC X(1)  VALUE '8'.
 01  PROCESS-NAME-CHAR            PIC X(1)  VALUE '*'.

 01  RETURN-CODES.
     05  RC-ZERO                  PIC 9(2)  VALUE  0.
/
 LINKAGE SECTION.

 COPY DKNHCMC0                     REPLACING ==:HCDM:==
                                   BY        ==HCDM==.
/
 PROCEDURE DIVISION                USING     HCDM-REC.

     PERFORM  VARYING SUB          FROM 1 BY 1
              UNTIL   SUB          > STRING-NAME-SIZE
         MOVE PROCESS-NAME-CHAR    TO HCDM-RECON-STG-BYTE (SUB)
     END-PERFORM.

     MOVE RECONCILIATION-ID        TO HCDM-RECON-STG-BYTE (1).

     MOVE RC-ZERO                  TO RETURN-CODE.

     GOBACK.
```

Figure 2-13. HCDM Exit 1 Routine Example 1

HCDM Exit 2 — Modify HCDM Reconciliation String Creation

This user exit allows you to:

- Inspect/modify each record before it is written to the reconciliation string.
- Insert reconciliation string records.
- Exclude records from being written to the reconciliation string.
- Initialize and maintain up to 16K of data from user-exit call to user-exit call.
- Deactivate the HCDM exit 2 for the remaining of HCDM processing.
- Terminate HCDM processing.

The HCDM exit 2 routine is called by HCDM before each write to the reconciliation string.

Note: Member HCMX002 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding COBOL HCDM user exit 2 routines.

Message HCDM 30030, which includes the exit routine name and the exit routine return code, is displayed on the CPCS-I supervisor's terminal and written to the scroll log when HCDM processing is terminated due to an exit 2 routine request (see HCDM messages in *CPCS-I Messages and Codes* for more information).

Activation

A different HCDM exit 2 routine may be specified for each financial institution. The exit is active when one or more HCDM exit 2 routine names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information). For record format and record profile information, see the chapter, "System and Application Profiles," in the *CPCS-I Customization Guide*.

No re-gens, CPCS-I restarts or CHAPs are required when new HCDM exit 2 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-16 (Page 1 of 2). HCDM Exit 2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters • This area contains the item-indicator that identifies the current record as 'matched', 'free' or 'missing'. • This area also contains a flag, indicating whether or not the user exit is being called for the first time. • It also contains a pointer to a 16K workspace allocated by HCDM for use by the exit.	DKNHCMC7	

Table 2-16 (Page 2 of 2). HCDM Exit 2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
HCDM Profile Parameters	DKNHCMC0	DKNHCMA0
Uncompressed Reconciliation Record	DKNCRZD	ZDDSCT
Uncompressed Master Record • This area contains binary zeros when the item-indicator identifies the reconciliation record as 'free'.	DKNCRZD	ZDDSCT
Uncompressed Process Record • This area contains binary zeros when the item-indicator identifies the reconciliation record as 'missing'.	DKNCRZD	ZDDSCT

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-17. HCDM Exit 2 Routine Return Codes

Code (dec)	Request
+00	Write the reconciliation record to the reconciliation string.
+04	Do not write the reconciliation record to the reconciliation string.
+08	Insert the reconciliation record into the reconciliation string, and pass the original reconciliation record again on the next exit routine call.
+12	Write the reconciliation record to the reconciliation string, and do not call the exit routine again for this run.
All others	Terminate HCDM processing.

Restrictions

The HCDM exit 2 routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.
- Should establish access to its 16K workspace by executing the following instruction:

  ```
  SET ADDRESS OF address to HCMX002-WS-POINTER
  ```

  Where:

  address Is a 01-level data name in the exit's LINKAGE section

The item-indicator must always be tested to determine what type of record is being passed to the exit routine. Accessing the master record buffer 'free' items, or accessing the process record buffer for 'missing' items, causes an 0C4 abend.

HCMX002-WS-POINTER must be tested for NULLS during program initialization. If it is NULLS, then HCDM failed to allocate a workspace. In this situation, do not attempt to use HCMX002-WS-POINTER in a 'SET ADDRESS OF' instruction, or try to access the workspace. If you do, the result is an S0C4 abend.

The user exit routine does not have access to the sort key, which is used to sort the reconciliation records into the original process string sequence. Therefore, duplicate record conditions occur when records are inserted.

The reconciliation output string is not created when HCDM is terminated due to an exit 2 request.

Example 1

Operation:

Use the HCDM exit 2 to:

1. Change ‘free’ item records with a user code of E2 to ‘matched’.
2. Exclude ‘free’ item records with a user code of E3 from being written to the output reconciliation string.
3. Insert a ‘free’ item record preceding each ‘matched’ item record with a user code of E4. The inserted record only differs from the current record in that it is flagged as a user control record of type 78.
4. Deactivate the HCDM exit 2 routine for the run if a ‘matched’ record with a user code of E5 is detected.
5. Terminate HCDM processing if a ‘free’ item with a user code of E6 is detected.

User Exit Routine:

```
  ID DIVISION.
  PROGRAM-ID. HCMX002.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  INSERT-PROCESSED-SWITCH        PIC X(1)  VALUE 'N'.
      88  INSERTED                            VALUE 'Y'.
      88  NOT-INSERTED                        VALUE 'N'.

  01  USER-CODES.
      05  UC-MATCH                   PIC X(1)  VALUE  X'E2'.
      05  UC-EXCLUDE                 PIC X(1)  VALUE  X'E3'.
      05  UC-INSERT                  PIC X(1)  VALUE  X'E4'.
      05  UC-DEACTIVATE              PIC X(1)  VALUE  X'E5'.
      05  UC-TERMINATE               PIC X(1)  VALUE  X'E6'.

  01  RETURN-CODES.
      05  RC-WRITE                   PIC 9(2)  VALUE  0.
      05  RC-EXCLUDE                 PIC 9(2)  VALUE  4.
      05  RC-INSERT                  PIC 9(2)  VALUE  8.
      05  RC-DEACTIVATE              PIC 9(2)  VALUE 12.
      05  RC-TERMINATE               PIC 9(2)  VALUE 16.
```

Figure 2-14 (Part 1 of 3). HCDM Exit 2 Routine Example 1

```
 01  USER-CONTROL                  PIC X(1)  VALUE  X'78'.
/
 LINKAGE SECTION.

 COPY DKNHCMC7.

 COPY DKNHCMC0                     REPLACING ==:HCDM:==
                                   BY        ==HCDM==.

 COPY DKNCRZD                      REPLACING ==:ZD:==
                                   BY        ZD-RECON.

 COPY DKNCRZD                      REPLACING ==:ZD:==
                                   BY        ZD-MASTER.

 COPY DKNCRZD                      REPLACING ==:ZD:==
                                   BY        ZD-PROCESS.
/
 PROCEDURE DIVISION                USING    HCMX002-USER-EXIT-PARMS
                                            HCDM-REC
                                            ZD-RECON
                                            ZD-MASTER
                                            ZD-PROCESS.

     MOVE RC-WRITE                          TO RETURN-CODE.

     EVALUATE TRUE

         WHEN II-FREE-ITEM
              EVALUATE ZD-RECON-USER-FLAGS

                  WHEN UC-MATCH
                       SET  II-MATCHED-ITEM  TO TRUE

                  WHEN UC-EXCLUDE
                       MOVE RC-EXCLUDE       TO RETURN-CODE

                  WHEN UC-TERMINATE
                       MOVE RC-TERMINATE     TO RETURN-CODE

              END-EVALUATE

         WHEN II-MATCHED-ITEM
              EVALUATE ZD-RECON-USER-FLAGS

                  WHEN UC-INSERT
                       IF  INSERTED
                           SET NOT-INSERTED  TO TRUE
                       ELSE
                           SET  II-FREE-ITEM
                                ZD-RECON-CONTROL TO TRUE
                           MOVE USER-CONTROL TO ZD-RECON-DOCU-TYPE
                           MOVE RC-INSERT    TO RETURN-CODE
                           SET  INSERTED     TO TRUE
                       END-IF
```

Figure 2-14 (Part 2 of 3). HCDM Exit 2 Routine Example 1

```
                WHEN UC-DEACTIVATE
                     MOVE RC-DEACTIVATE    TO RETURN-CODE

            END-EVALUATE

      END-EVALUATE

      GOBACK.
```

Figure 2-14 (Part 3 of 3). HCDM Exit 2 Routine Example 1

HCDM Exit 3 — Modify HCDM Unmatched Item Processing

This user exit may be used to rematch unmatched process string records under a different match criteria.

The HCDM exit 3 is entered whenever a process string record fails to match any master string record, in both standard and positional matches. Upon return, if any of the profile match indicators have been modified by the exit routine, HCDM rematches the process record under the new criteria. If the match fails on a subsequent attempt, HCDM does not call the exit 3 again for the unmatched process record. The original match criteria is restored after the rematch.

Note: Member HCMX003 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding COBOL HCDM exit 3 routines.

Activation

A different HCDM exit 3 routine may be specified for each financial institution. The exit routine is active when one or more HCDM exit 3 routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information). For record format and record profile information, see the chapter, “System and Application Profiles,” in the *CPCS-I Customization Guide*.

No re-gens, CPCS-I restarts or CHAPs are required when new HDCM exit 3 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-18. HCDM Exit 3 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Uncompressed Unmatched Process Record	DKNCRZD	ZDDSCT
HCDM Profile Parameters	DKNHCMC0	DKNHCMA0

Restrictions

The HCDM exit 3 routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the HCDM exit 3 to rematch unmatched process records with a user code of E1. The same match criteria used in the first match, without matching on field 2, is used on the subsequent match.

User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. HCMX003.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  USER-CODES.
     05  UC-REMATCH                PIC X(1)  VALUE  X'E1'.

 01  RETURN-CODES.
     05  RC-ZERO                   PIC 9(2)  VALUE  0.
/
 LINKAGE SECTION.

 COPY DKNCRZD                       REPLACING ==:ZD:==
                                    BY        ==ZD==.

 COPY DKNHCMC0                      REPLACING ==:HCDM:==
                                    BY        ==HCDM==.
/
 PROCEDURE DIVISION                 USING     ZD
                                              HCDM-REC.

     IF  ZD-USER-FLAGS                        =  UC-REMATCH
         MOVE ' '                             TO HCDM-FIELD02.

     MOVE RC-ZERO                             TO RETURN-CODE.

     GOBACK.
```

Figure 2-15. HCDM Exit 3 Routine Example 1

HCDM Exit 4 — Modify HCDM Positional Match Sort Criteria

This user exit only applies to positional matches. It allows to:

- Inspect/modify the sort criteria.
- Terminate HCDM processing.

The HCDM exit 4 is entered whenever a positional match is going to be performed, before the sort takes place. The exit routine may then temporarily modify the HCDM profile sort-field sequence, or stop the run.

Note: Member HCMX004 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding COBOL HCDM exit 4 routines.

Message HCDM 39023, which includes the exit routine name and the exit routine return code, is displayed on the CPCS-I supervisor's terminal and written to the scroll log when HCDM is terminated due to an exit 4 routine request (see HCDM messages in *CPCS-I Messages and Codes* for more information).

Activation

A different HCDM exit 4 routine may be specified for each financial institution. The exit is active when one or more HCDM exit 4 routine names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when new HCDM exit 4 routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-19. HCDM Exit 4 Communication Control Block

Control Block	COBOL Copybook	Assembler Macro/ Copybook
HCDM Profile Parameters	DKNHCMC0	DKNHCMA0

The HCDM profile parameters are passed to the exit routine. COBOL routines may include copybook DKNHCMC0 to reference this control block. Assembler routines may include copybook DKNHCMAC0 to map it.

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-20. HCDM Exit 4 Routine Return Codes

Code (dec)	Request
+00	Resume HCDM processing.
All others	Terminate HCDM processing.

Restrictions

The HCDM exit 4 routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the HCDM exit 4 to:

1. Stop HCDM execution if a positional match is to be performed and only one of the two strings must be sorted.
2. Overlay the sort sequence of any valid positional sort, specifying the account number and amount fields.

User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. HCMX004.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  SORT-SEQUENCE                 PIC X(16) VALUE '31'.
 01  SORT-ON                       PIC X(01) VALUE '1'.

 01  RETURN-CODES.
     05  RC-PROCEED                PIC 9(02) VALUE  0.
     05  RC-TERMINATE              PIC 9(02) VALUE  4.
/
 LINKAGE SECTION.

 COPY DKNHCMC0                      REPLACING ==:HCDM:==
                                    BY        ==HCDM==.
/
 PROCEDURE DIVISION                 USING     HCDM-REC.

     IF    HCDM-SORT-PROCESS        =  SORT-ON
       AND HCDM-SORT-MASTER         =  SORT-ON
           MOVE SORT-SEQUENCE       TO HCDM-SORT-FLD-SEQ
           MOVE RC-PROCEED          TO RETURN-CODE
     ELSE
           IF  HCDM-SORT-PROCESS    =  SORT-ON
            OR HCDM-SORT-MASTER     =  SORT-ON
               MOVE RC-TERMINATE    TO RETURN-CODE.

     GOBACK.
```

Figure 2-16. HCDM Exit 4 Routine Example 1

IDCR Exit — Access Adjustment Descriptions

This exit allows the access of adjustment code descriptions for inwork DCV reconciliation reporting.

Notes:

1. Member DKNDADJX of CPCSI.V01R01.SDKNSAM2 may be used as a reference when coding COBOL IDCR exit routines.
2. The same user routine used for the IDCR exit, is also used for the ODCR exit. Also, the IDCR and ODCR exits are activated simultaneously.

Activation

A different IDCR exit routine may be specified for each financial institution. The exit is active when one or more IDCR exit routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when new IDCR exit routine versions are installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-21. IDCR Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
Call Parameters	DKNCADJX	

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-22. IDCR Exit Routine Return Codes

Code (dec)	Description
+00	Request completed
+39	Adjustment code not on file
+40	Process error
+41	Invalid function code

Restrictions

The IDCR exit routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the IDCR exit to retrieve adjustment code descriptions from a VSAM data set.

User Exit Routine:

Note: Member DKNDADJX of CPCSI.V01R01.SDKNSAM2 contains the COBOL code for this example.

MDIS Exit — Modify M-string Distribution

The MDIS user exit receives each codeline record being processed, once MDIS has made a distribution decision about the record. The exit allows you to:

- Distribute codeline records to D-strings other than the ones selected by MDIS.
- Exclude codeline records from being distributed.
- Change the pocket number of codeline records.
- Distribute non-distributable records.
- Deactivate the exit routine for the remaining MDIS processing.
- Terminate MDIS processing.
- Build a table of paper user-control documents. These documents are marked for distribution by DKNMDIS.
- Set a switch that tells MDIS the input string is not from the Balancing feature of the High Performance Transaction System.

The exit routine can perform first or last call processing based on the value of a passed call-sequence indicator.

The string directory UD flag is set for each MDIS distributed D-string that has been altered by the exit routine.

The 'distribute non-distributable record' request is satisfied without affecting the current status of the MDIS record save buffers.

No output is generated when the 'terminate MDIS' option is requested (distributed records are only written when all the input has been processed).

Message MDIS 30013, which includes the exit routine name, is displayed on the CPCS-I supervisor's terminal and written to the scroll log when MDIS processing is terminated due to an exit routine request (see MDIS messages in *CPCS-I Messages and Codes* for more information).

Note: Members DKNMDIX and DKNMDIX2 of CPCSI.V01R01.SDKNSAM2 may be used as templates when coding MDIS user exit routines in assembler and COBOL respectively.

Activation

A different MDIS exit routine can be specified for each financial institution and sort type. The exit is active when one or more MDIS exit routine names are specified in the bank control file, or when an MDIS sxit routine name is specified in the sort pattern definition for the distribution, with the sort pattern definition taking precedence over the bank control file when both specify exit routines (see "Bank Control File Record Formats" and "Sort-Pattern-Definition Record Formats" in the *CPCS-I Customization Guide* for more information).

A CPCS-I system-wide MDIS user exit routine can also be activated by specifying its name in the MDIS DKNBLDL entry via the USREXIT parameter. An MDIS exit routine specified in the DKNBLDL table gets activated only when the bank control file and sort pattern definition do not specify any MDIS exit routine for the distribution. See "Describing an Application Task's Environment" in the *CPCS-I Customization Guide* for more information on the USREXIT parameter.

No re-gens, CPCS-I restarts or CHAPs are required when MDIS exits are activated/deactivated via the bank control file or sort pattern definitions, or when new MDIS exit routine versions are installed. Module DKNBLDL must be re-assembled and CPCS-I must be restarted when MDIS exits are activated/deactivated via the USREXIT parameter.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-23. MDIS Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Call Parameters • MDIS processing modification requests are made via the reason-code field in this control block.	DKNMDIXC	DKNMDIXP
Uncompressed Codeline Record	DKNCRZD	ZDDSCT
String's Expanded Directory Record Description (ZB)	DKNCRZB	ZBDSCT
MDIS Candidate List • This list contains the pockets that are valid for distribution.	DKNMDISS	

Restrictions

MDIS exit routines should preferably be written in COBOL, but may also be written in assembler.

The candidate list should not be modified by the MDIS exit routine.

Codeline records that have their pocket number or distribution pocket number changed to an invalid (not in the candidate list) pocket number, do not get distributed.

Codeline records that have their pocket number changed, get distributed to the D-string corresponding to the new pocket number.

Example 1

Operation:

Use the MDIS user exit routine to:

1. Stop the M-string distribution if a record with a user-code greater than F0 is detected.
2. Deactivate the user exit routine for the remaining process if a record with a user code of E1 is detected.
3. Exclude records with a user code of E0 from being distributed.
4. Distribute all records with a user code of E2 to the D-string for pocket 06.
5. Change the pocket number of all records with a user code of E3 to pocket 03.

User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. MDISEXT1.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  USER-CODES.
     05  UC-EXCLUDE                  PIC X(1) VALUE X'E0'.
     05  UC-DEACTIVATE               PIC X(1) VALUE X'E1'.
     05  UC-CHANGE-DISTRIBUTION      PIC X(1) VALUE X'E2'.
     05  UC-CHANGE-POCKET            PIC X(1) VALUE X'E3'.
     05  UC-LIMIT                    PIC X(1) VALUE X'F0'.
     05  UC-MAXIMUM                  PIC X(1) VALUE X'FF'.

 01 POCKET-CODES.
     05  PC-NEW-POCKET               PIC X(2) VALUE '03'.
     05  PC-NEW-DIST                 PIC X(2) VALUE '06'.
/
 LINKAGE SECTION.

 COPY DKNMDIXC                        REPLACING ==:TAG:==
                                      BY        ==MDIX==.

 COPY DKNCRZD                         REPLACING ==:ZD:==
                                      BY        ==ZD==.

 COPY DKNCRZB.

 COPY DKNMDISS.
/
 PROCEDURE DIVISION                   USING MDIX-CALL-PARMS
                                            ZD
                                            ZB
                                            MDIS-CANDIDATE-LIST.

     SET MDIX-CURRENT-POCKET                TO TRUE.

     IF  MDIX-DISTRIBUTE

         EVALUATE TRUE

             WHEN UC-LIMIT THRU UC-MAXIMUM
                  SET MDIX-TERMINATE-MDIS   TO TRUE

             WHEN UC-DEACTIVATE
                  SET MDIX-NO-MORE-CALLS    TO TRUE

             WHEN UC-EXCLUDE
                  SET MDIX-DO-NOT-DIST      TO TRUE

             WHEN UC-CHANGE-DISTRIBUTION
                  MOVE PC-NEW-DIST          TO MDIX-DIPKT
                  SET MDIX-MOVE-TO-DIPKT    TO TRUE
```

Figure 2-17 (Part 1 of 2). MDIS Exit Routine Example 1

```
            WHEN UC-CHANGE-POCKET
                 MOVE PC-NEW-POCKET         TO MDIX-DIPKT
                 SET MDIX-CHANGE-DIPKT      TO TRUE

        END-EVALUATE.

    GOBACK.
```

Figure 2-17 (Part 2 of 2). MDIS Exit Routine Example 1

Example 2

Operation:

Use the MDIS user exit routine to force the distribution of original reject control records (which are not normally distributed).

User Exit Routine:

Note: Member DKNDADJX, in the CPCSI.V01R01.SDKNSAM2, contains the COBOL code for this example.

MICR Exit 1 — Customize MICR Entry Begin Parameter Validations

This exit permits additional validation of the operator-entered MICR BEGIN parameters. It determines whether to return a diagnostic message to the operator or to permit the capture to start. If specified, the exit routine is entered after the operator presses **ENTER** to verify the BEGIN screen parameters, before the capture starts.

When accepting an entry, the exit might cause 4 bytes of user data, for use by application programs, to go into the string-header record of the output I-string. The MICR task places this information in field DISUSPA (DIDSCT DSECT) of the string-header record.

In addition to the normal BEGIN screen input, the user exit routine can inspect up to 12 characters, entered on the OPTNS line of the BEGIN screen, and process accordingly, based on user criteria. The first two of these 12 characters must be 'U='. The rest of the field is user-dependent, and gets its lowest order bytes padded with binary zeros if the data entered is less than 12 characters. The 12 bytes get filled with blanks when the parameter is omitted. For more information about the OPTNS parameter, see the *CPCS-I Terminal Operations Guide*.

For a more extensive processing by the user exit routine, a control block that can be mapped by the BEGNDSCT DSECT can be used to inspect all the information entered on the BEGIN screen.

If the user exit routine determines that the data entered is wrong, it returns a 32-byte message displayed on the BEGIN screen. The displayed message gets identified as USEX1031. If the operator does not then enter the correction that the message specifies, the capture does not start.

The MICR task does not allow prime pass captures when the end-prime status is active. These restriction may be overridden with a MICR BEGIN exit routine, setting the DATEPOVR flag in the DATAEPRM field (BEGNDSCT DSECT). Upon return from the user exit routine, DKNMICR sends message MBEG2096 to the scroll log if that flag is on. Otherwise, another message, indicating that a prime capture is not allowed, appears and MICR processing stops (see "MICR BEGIN Error Messages", in *CPCS-I Messages and Codes*, and the *CPCS Enhanced System Manager User's Guide* for more information.

Note: You can use sample DKNMXS11 as a template when you code a MICR exit 1 user routine.

Activation

You can specify a different MICR exit routine 1 for each financial institution. The exit is active when one or more MICR exit 1 routine names are specified in the bank control file ("Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when the exit is activated/deactivated or when a new exit routine version is installed. The exit routine is dynamically loaded by the MICR BEGIN process and deleted by the MICR END process.

Linkage

The standard OS/390 linkage conventions are followed.

Information is passed in the following control block:

Table 2-24. MICR Exit 1 Communication Control Block

Control Block	Assembler Macro/ Copybook
Call Parameters	DLMMEX11

The private work area address field points to a 300 byte work area available to the user exit routine. The area is initialized to binary zeroes before the routine is entered for the first time (this also applies to capture restarts) in a capture. Its contents are preserved from call to call during the rest of the capture.

The user exit routine is responsible for setting the return code, indicating whether the entry begin parameters are considered valid or not.

The diagnostic message must be returned when the parameter validations fail.

If the four bytes of data are returned on a successful validation, they will be included in the string's header record.

The MICR begin work area can be mapped by the BEGNDSCT copybook.

Restrictions

The MICR exit 1 must be active when the 'U=' parameter is entered on the MICR Begin screen. MICR exit 1 routines:

- Must be written in assembler
- Must be reentrant
- Must not include any supervisor or data management macro instructions. Otherwise unpredictable results might occur.

Example 1

Operation:

Make a two-digit pass type a required user parameter, and if valid (01-06), save it into the string-header record.

User Exit Routine:

```
MBGNEXT1 CSECT
MBGNEXT1 AMODE 24
MBGNEXT1 RMODE 24
         SAVE  (14,12)               SAVE REGISTERS
         BASR  12,0                  GET ROUTINE @
         USING *,12                  BASE ROUTINE
*
         L     R2,0(R1)              GET INTERFACE CNTL BLOCK @
         USING M1INCNTL,R2           MAP INTERFACE CNTL BLOCK
         ICM   R3,B'1111',M1WAREA@   GET WORK AREA @
         USING PWRK,R3               MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         CLC   M1BOPTNS(2),PARMID    USER PARM ENTERED ?
         BNE   SETERR                N. INVALID PARMS
*
         CLC   M1BOPTNS+2(2),LOWLIMIT PARM<LOWER LIMIT?
          BL    SETERR                Y. INVALID PARMS
*
         CLC   M1BOPTNS+2(2),UPPLIMIT PARM>UPPER LIMIT?
         BH    SETERR                Y. INVALID PARMS
*
         MVC   M1STGHDD(2),M1BOPTNS+2 SET STG HDR DATA
         B     RETURN                EXIT ROUTINE
*
SETERR   MVI   M1RETCDE,INVALRC      SET ERROR RETURN CODE
         MVC   M1DIGMSG,ERRMSG       SET ERROR MESSAGE
*
RETURN   L     R13,4(,R13)           REST CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    EXIT PROGRAM
*
         DKNREGS                     REGISTER EQUATES
*
INVALRC  EQU   X'04'                 ERROR RETURN CODE
PARMID   DC    C'U='                 PARM IDENTIFIER
LOWLIMIT DC    C'01'                 PARM RANGE LOWER LIMIT
UPPLIMIT DC    C'06'                 PARM RANGE UPPER LIMIT
ERRMSG   DC    CL32'MBGNEXT1 - ENTER VALID PASS TYPE'
*
PWRK     DSECT                       PRIVATE WORK AREA
SAVEAREA DS    18F                   OUR SAVE AREA
*
         DKNMEX11                    COMMUNICATIONS CNTL BLOCK
*
         END
```

Figure 2-18. MICR BEGIN Exit Routine Example 1

Example 2

Operation:

Ensure sort type 001 is only run for cycle 8, allowing prime pass captures even if end-prime is active.

User Exit Routine:

```
MBGNEXT2 CSECT
MBGNEXT2 AMODE 24
MBGNEXT2 RMODE 24
         SAVE  (14,12)               SAVE REGISTERS
         BASR  12,0                  GET ROUTINE @
         USING *,12                  BASE ROUTINE
*
         L     R2,0(R1)              GET INTERFACE CNTL BLOCK @
         USING M1INCNTL,R2           MAP INTERFACE CNTL BLOCK
         ICM   R3,B'1111',M1WAREA@   GET WORK AREA @
         USING PWRK,R3               MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         ICM   R5,B'1111',M1BGWRK@   GET BEGIN WRK AREA@
         USING WKAREA,R5             MAP MICR BEGIN WORK AREA
*
         CLC   BGNTYPE,TYPE001       RESTRICTED SORT TYPE ?
         BNE   EXITOK                N. VALID PARMS
*
         CLC   BGNCYCLE,CYCLE8       VALID CYCLE ?
         BNE   SETERR                N. INVALID PARMS
*
         CLC   BGNPAS,PRPASS         PRIME PASS ?
         BNE   EXITOK                N. VALID PARMS
*
         TM    DATAEPRM,DATEPACT     END PRIME ACTIVE ?
         BNO   EXITOK                N. VALID PARMS
         OI    DATAEPRM,DATEPOVR     Y. ALLOW CAPTURE
         B     EXITOK                N. EXIT ROUTINE
*
SETERR   MVI   M1RETCDE,INVALRC      SET ERROR RETURN CODE
         MVC   M1DIGMSG,ERRMSG       SET ERROR MESSAGE
         B     RETURN                EXIT ROUTINE
*
EXITOK   MVI   M1RETCDE,M1VALPMS     SET SUCCESS RET CODE
         XC    M1STGHDD,M1STGHDD     CLEAR STG HDR DATA
*
RETURN   L     R13,4(,R13)           REST CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    EXIT PROGRAM
*
         DKNREGS                     REGISTER EQUATES
*
```

Figure 2-19 (Part 1 of 2). MICR BEGIN Exit Routine Example 2

```
CYCLE8   DC    C'8'                   VALID CYCLE
TYPE001  DC    X'01'                  RESTRICTED SORT TYPE
PRPASSIT DC    C'1'                   PRIME PASS TYPE
INVALRC  DC    X'04'                  ERROR RETURN CODE
ERRMSG   DC    CL32'MBGNEXT2 - ENTER VALID CYCLE'
*
PWRK     DSECT                        PRIVATE WORK AREA
SAVEAREA DS    18F                    OUR SAVE AREA
*
         DKNMEX11                     COMMUNICATIONS CNTL BLOCK
*
         END
```

Figure 2-19 (Part 2 of 2). MICR BEGIN Exit Routine Example 2

MICR Exit 2 — Customize MICR Document Processing

This exit permits additional user processing over and above that which the user stacker-select routine has already performed in the document processor. Such additional processing might include:

- Additional item validation that was not necessary to determine pocket selection.
- Analysis of rejected items and setting of communications flags to affect processing of the item when received by the user-edit routine during online reject-reentry.
- Determination of document sequence errors that should cause entry ending or restart or other operator intervention. For example, it could check for too many rejects in a row or for control documents that are not valid for the type of run.
- Preparation of user-defined control records for insertion into the output I-string.
- Checking for incorrect programmable endorse (XP document processors only).
- Insertion of user data into expanded codeline record fields.

If the MICR document processing user exit routine is active, it receives the codeline records (see “Restrictions” on page 2-61), after they have been processed by the document processor and reformatted, and before they are processed by the MICR task and written to the output I-string. In addition, the routine is also entered:

- before the first document (the string-header record is passed in this case)
- before each kill bundle control record
- if data items were captured, after the last document (the end of string dummy record is passed in this case) completes processing

You can use the user exit routine to inspect and change records, insert records and stop captures.

Inspecting/Changing Records

The codeline data and control information of a passed record can be inspected by the exit routine; however only the user flag byte, flag 3 bytes 4 and 5, and expanded fields should be changed.

Inserting Records

The MICR task validates fields DIFLAG2, DITYPEI, and DIPKT (see “Restrictions” on page 2-61), processes the insertion record, and writes it before processing current record.

If a record must be inserted after a specific document, the user exit routine must make the request when the next document is obtained.

Several contiguous records can be inserted before a given document. The Insert and Return codes cause a repetitive call to the exit routine (the user record is written but the current document is passed to the exit routine without being written). A loop of repetitive calls can be generated using this technique to insert multiple records.

Stopping Captures

A MICR document processing exit routine can request to stop the document processor operations and communicate with the MICR operator. If this is done, no more document processing is attempted, and the operator must either end or suspend the capture. Operator communications consist of displaying message 023 along with an exit routine supplied message on the MICR STATUS screen (see "MICR Status Messages", in *CPCS-I Messages and Codes* for more information). The MICR task does not honor stop requests in some particular cases (see "Restrictions" on page 2-61).

The exit routine decides if the current record being processed is written to the output I-string or not. Records following this record are not written to the output I-string.

Note: You can use the sample DKNMXS21 as a template when you code MICR exit 2 user routines.

Activation

A different MICR document processing exit routine may be specified for each financial institution and sort type. The exit is active when one or more MICR exit 2 routine names are specified in the bank control file, or when a MICR exit 2 routine name is specified in the sort pattern definition for a capture, with the sort pattern definition taking precedence over the bank control file when both specify exit routine names (see "Bank Control File Record Formats" and "Sort-Pattern-Definition Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when the exit routine is activated/deactivated or when new exit routine versions are installed. The exit routine is dynamically loaded by the MICR BEGIN process and deleted by the MICR END process.

Linkage

The standard OS/390 linkage conventions are followed.

Information is passed in the following control block:

Table 2-25. MICR Exit 2 Communication Control Block

Control Block	Assembler Macro/ Copybook
Call Parameters	DKNMEX21

The private work area address field points to a 300-byte work area available to the user exit routine. The area is initialized to binary zeroes before the routine is entered for the first time (this also applies to capture restarts) in a capture. Its contents are preserved from call to call during the rest of the capture.

The exit routine is responsible for setting the return code, requesting a function to be performed by the MICR task.

The diagnostic message must be returned when the 'Stop Capture' code is returned.

The new codeline record address field must be returned when the 'Replace Record' or 'Insert Record' codes are returned.

Restrictions

MICR exit 2 routines:

- Must be written in assembler.
- Must be reentrant.

The exit routine should ***not***:

- Include any supervisor or data management macro instructions. If it does, unpredictable results might occur.
- Perform any kind of data set I/O. If such a process is needed, it should be implemented as a post-capture task that would sequentially process the output I-string.
- Branch to the MICR task internal dispatcher.

Codeline records corresponding to divider documents are not passed to the exit routine. Codeline records corresponding to tracers are not passed to the exit routine, with the exception of the first tracer slip of each tracer group (none of the tracer slips of non-entry HSRR tracer groups are passed to the user exit). Tracer slip numbers should not be changed; however, records can be inserted before a tracer slip record is written to the I-string.

Unpredictable results can occur if portions of the record other than the user flag byte, flag 3 bytes 4 and 5, and expanded fields (9 through 15) are changed by the exit routine.

Inserted records must have bit 0 on (which identifies them as control records) in field DIFLAG2 and any code from X'00' to X'7F' (which identifies them as user records) in field DITYPEI. The pocket number must be valid for the document processor on which the capture takes place. Also, the first byte must not be X'FF' (which identifies the end of string dummy record). On return from the user exit routine, the MICR task turns on the highest order bit of field DIFLAG2 if it is not on, turns off the highest order bit of field DITYPEI if it has a value greater than X'7F', and overlays the pocket number with the system reject pocket number if it is invalid.

Insertions are not allowed on the first (initialization) exit routine call, because the first record in the string must always be the string-header record.

Messages associated to stop requests can be up to 40 characters long, and can contain any EBCDIC characters except % and ?.

Initialization (no documents have been captured yet) and termination (the capture has already been ended/suspended by the operator) stop requests are not issued by MICR.

Note: The exit (if active) is also entered on capture restarts during the reprocessing of previously captured items. You can identify this condition by interrogating the RSCB control block RSCBRST flag.

The following RSCB control block flags allow the routine to identify different manual and automatic (enhanced prime runs) “End of Entry” situations:

RSCBPAUS
Data items were captured.

RSCBCANC
Entry is suspended (CA was entered on the MICR status screen and data items are captured for the current entry).

RSCBEPLA
Enhanced prime pass ended (E was entered on the MICR status screen) and data items are captured for the current entry.

The exit routine is not invoked when an entry is cancelled (CA was entered on the MICR status screen and no data items are captured for the current entry). The user exit routine request code is ignored if the entry is being ended.

Example 1

Operation:

Use the MICR document processing exit to:

1. Verify that the first document following a tracer group is always a block ticket on prime passes.
2. Verify that a block ticket is always followed by a batch ticket and vice-versa on prime passes.
3. Stop the capture if 20 or more consecutive rejects are obtained.
4. Stop the capture if 20 or more consecutive item numbering errors occur.
5. Stop the capture if 20 or more consecutive endorsing errors occur.

User Exit Routine:

```
MDPREXT1 CSECT
MDPREXT1 AMODE 24
MDPREXT1 RMODE 24
         SAVE  (14,12)               SAVE REGISTERS
         BASR  R12,0                 GET ROUTINE @
         USING *,R12                 MAP ROUTINE
*
         L     R2,0(R1)              GET INTRF CNTL BLK @
         USING M2INCNTL,R2           MAP INTRF CNTL BLK
         ICM   R3,B'1111',M2WAREA@   GET PRIV WRK AREA @
         USING PWRK,R3               MAP PRIVate work area
*
         LA    R14,SAVEAREA          POINT TO OUR SAVE AREA
         ST    R14,8(R13)            STORE FORWARD SAVE AREA @
         ST    R13,4(R14)            STORE BACKWARD SAVE AREA @
         LR    R13,R14               POINT TO NEW SAVE AREA
*
         ICM   R4,B'1111',M2CDELN@   GET CURRENT REC @
         USING DIDSCT,R4             MAP CURRENT CODELINE REC
         ICM   R5,B'1111',M2MUPA@    GET MUPA @
         USING MUPADSCT,R5           MAP MUPA
*
         CLC   DI(10),DUMMREC        END OF RUN ?
         BE    END                   Y. RETURN WITH NO REQUEST
         TM    DIFLAG2,DIMTCR        CONTROL RECORD ?
         BZ    NONCTL                N. SKIP CNTL REC LOGIC
         CLI   DITYPEI,DIBK          BLOCK SLIP RECORD ?
         BE    BLOCK                 Y. PROCESS BLK SLIP REC
         CLI   DITYPEI,DIBS          BATCH SLIP RECORD ?
         BE    BATCH                 Y. PROCESS BATCH SLIP REC
         CLI   DITYPEI,DISTGHD       1ST EXIT ROUTINE CALL ?
         BE    INITIAL               Y. DO INITIALIZATION
         B     RESUME                N. RETURN WITH NO REQUEST
*
BLOCK    OI    RSWITCH,RSWBLK        SET BLK DOC PROCESSED
         B     RESUME                RETURN WITH NO REQUEST
*
BATCH    TM    RSWITCH,RSWFST        1ST DOCUMENT PROCESSED ?
         BO    RESETB                Y. RESET BLOCK SWITCH
         TM    RSWITCH,RSWBLK        BLK SLIP PRECEDING DOC ?
         BZ    MESSAGE1              N. SEQUENCE ERROR
         OI    RSWITCH,RSWFST        SET FIRST DOC SWITCH
*
RESETB   NI    RSWITCH,X'FF'-RSWBLK  RESET BLK SLIP SW
         B     RESUME                RETURN WITH NO REQUEST
*
```

Figure 2-20 (Part 1 of 3). MICR Document Processing Exit Routine Example 1

```
NONCTL   TM    RSWITCH,RSWFST         DOCUMENT SEQUENCE OK ?
         BZ    MESSAGE1               N. SEQUENCE ERROR
         TM    RSWITCH,RSWBLK         PRECEDING DOC A BLK ?
         BO    MESSAGE2               Y. BATCH SLIP MISSING
         CLI   DIPKT,DIREJ            REJECTED DOCUMENT ?
         BNE   NORJCT                 N. RESET ERROR COUNTER
         LH    R10,RSCTR              Y. INCREMENT
         LA    R10,1(R10)             ERROR
         STH   R10,RSCTR              COUNTER
         CH    R10,MAXREJ             EXCESSIVE REJECTS ?
         BL    CKINF                  N. CHECK FOR INF VALIDITY
         B     MESSAGE3               Y. PRODUCE ERROR MESSAGE
 *
NORJCT   XC    RSCTR,RSCTR            CLEAR REJECT COUNTER
*
CKINF    TM    DI3BYTE1,PINFINV       INF OPERATN INVALID ?
         BNO   NOINFERR               N. RESET INF ERROR COUNTER
         LH    R10,RSINF              INCREMENT
         LA    R10,1(R10)             INF ERROR
         STH   R10,RSINF              COUNTER
         CH    R10,MAXREJ             EXCESSIVE INF REJECTS ?
         BL    CKEND                  N. CHECK FOR ENDORSE VALID
         B     MESSAGE4               PRODUCE INF ERROR MESSAGE
*
NOINFERR XC    RSINF,RSINF            CLEAR INF REJECT COUNTER
*
CKEND    TM    DI3BYTE1,PEDRSINV      ENDORSE INVALID ?
         BNO   NOENDERR               N. RESET ENDORSE ERR CNT
         LH    R10,RSENDR             INCREMENT
         LA    R10,1(R10)             ENDORSE ERROR
         STH   R10,RSENDR             COUNTER
         CH    R10,MAXREJ             EXCESSIVE ENDORSE REJECTS?
         BL    RESUME                 N. RETURN WITH NO REQUEST
         B     MESSAGE5               PRODUCE ENDORSE ERROR MSG
*
NOENDERR XC    RSENDR,RSENDR          CLEAR ENDORSE REJECT CNTR
         B     RESUME                 RETURN TO MICR
*
INITIAL  XC    RSA,RSA                CLEAR THE WORK AREA
         CLI   MUPAPPH,X'01'          PRIME(X'01')/HSRR(X'00') ?
         BNH   RESUME                 Y. RETURN WITH NO REQUEST
         OI    RSWITCH,RSWFST         SET SWITCH FOR BYPASSING
*                                     BLOCK/BATCH SEQ CHECKING
*                                     ON SUBSEQUENT PASS
         B     RESUME                 RETURN WITH NO REQUEST
*
END      XC    RSA,RSA                CLEAR THE WORK AREA
         B     RESUME                 RETURN WITH NO REQUEST
*
MESSAGE1 MVC   M2DIGMSG,MSG1          SET ERROR MESSAGE
         B     ERRTRN                 CONTINUE
*
MESSAGE2 MVC   M2DIGMSG,MSG2          SET ERROR MESSAGE
         B     ERRTRN                 CONTINUE
*
```

Figure 2-20 (Part 2 of 3). MICR Document Processing Exit Routine Example 1

```
MESSAGE3 MVC   M2DIGMSG,MSG3          SET ERROR MESSAGE
         B     ERRTRN                 CONTINUE
*
MESSAGE4 MVC   M2DIGMSG,MSG4          SET ERROR MESSAGE
         B     ERRTRN                 CONTINUE
*
MESSAGE5 MVC   M2DIGMSG,MSG5          SET ERROR MESSAGE
*
ERRTRN   MVI   M2RETCDE,M2STOPCP      SET 'STOP' RET CODE
         B     EXITPR                 EXIT PROGRAM
*
RESUME   MVI   M2RETCDE,M2CONTPR      SET 'RESUME' RET CODE
*
EXITPR   L     R13,4(,R13)            REST CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN TO MICR
*
DUMMREC  DC  10X'FF'                  END OF STRING DUMMY REC ID
MAXREJ   DC    H'20'                  CONSECUTIVE REJECT LIMIT
*
MSG1     DC CL40'BLOCK SEQUENCE ERROR - CANCEL/RESTART'
MSG2     DC CL40'BATCH SEQUENCE ERROR - CANCEL/RESTART'
MSG3     DC CL40'EXCESSIVE REJECTS - CANCEL/RESTART'
MSG4     DC CL40'EXCESSIVE INF ERRORS - CANCEL/RESTART'
MSG5     DC CL40'EXCESSIVE ENDR ERRORS - CANCEL/RESTRT'
*
         DKNREGS                      REGISTER EQUATES
*
PRMHSRR  EQU   X'01'                   PRIME/HSRR FLAG
*
PWRK     DSECT                        PRIVATE WORK AREA
SAVEAREA DS  18F                      OUR SAVE AREA
RSA      DS   0XL7                    AGGREGATE LENGTH ATTRIBUTE
RSCTR    DS    H                      REJECT COUNTER
RSINF    DS    H                      INF REJECT COUNTER
RSENDR   DS    H                      ENDORSE REJECT COUNTER
RSWITCH  DS    X                      PROGRAM SEQUENCE SWITCHES
RSWBLK   EQU   X'40'                  BLOCK TICKET PROCESSED
RSWFST   EQU   X'80'                  BYPASS RECORD
*
DIDSCT   DSECT                        MDS FORMATTED RECORD
         COPY  DIDSCT
*
MUPADSCT DSECT                        MICR USER PARAMETER AREA
         COPY  MUPA
*
         DKNMEX21                     MICR EXIT 2 INTRF CNTL BLK
*
         END
```

Figure 2-20 (Part 3 of 3). MICR Document Processing Exit Routine Example 1

Example 2

Operation:

Use the MICR document processing exit to:

1. Identify block ticket records with the characters BLOCK in field 15.
2. Identify batch ticket records with the characters BATCH in field 15.

3. Stop the capture if 500 or more consecutive rejects are obtained.
4. Process credit document records as follows:
   - Identify them with the characters CREDIT in field 15.
   - Place the auxiliary on-us data in field 12.
   - Place the on-us data in field 14.
5. Process debit document records and identify them with the character DEBIT in field 15.

User Exit Routine:

```
MDPREXT2 CSECT
MDPREXT2 AMODE 24
MDPREXT2 RMODE 24
         SAVE  (14,12)                  SAVE REGISTERS
         BASR  R12,0                    GET ROUTINE @
         USING *,R12                    MAP ROUTINE
*
         L     R2,0(R1)                 GET INTERF CTL BLOCK @
         USING M2INCNTL,R2              MAP INTERF CNTL BLOCK
         ICM   R3,B'1111',M2WAREA@      GET PRIV WRK AREA @
         USING PWRK,R3                  MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA             POINT TO OUR SAVE AREA
         ST    R14,8(R13)               STORE FORWARD SAVE AREA @
         ST    R13,4(R14)               STORE BACKWARD SAVE AREA @
         LR    R13,R14                  POINT TO NEW SAVE AREA
*
         ICM   R4,B'1111',M2CDELN@      GET CURR REC @
         USING DIDSCT,R4                MAP CURRENT CODELINE REC
*
         CLC   DI(10),DUMMREC           END OF RUN ?
         BE    END                      Y. RETURN WITH NO REQUEST
         TM    DIFLAG2,DIMTCR           CONTROL RECORD ?
         BZ    NONCTL                   N. SKIP CONTROL REC LOGIC
         CLI   DITYPEI,DIBK             BLOCK SLIP RECORD ?
         BE    BLOCK                    Y. PROCESS BLOCK SLIP REC
         CLI   DITYPEI,DIBS             BATCH SLIP RECORD ?
         BE    BATCH                    Y. PROCESS BATCH SLIP REC
         B     RESUME                   RETURN WITH NO REQUEST
*
BLOCK    MVC   DIFLD15(8),BLKID         SET BLOCK IDENTIFIER
         B     RESUME                   RETURN WITH NO REQUEST
*
BATCH    MVC   DIFLD15(8),BTHID         SET BATCH IDENTIFIER
         B     RESUME                   RETURN WITH NO REQUEST
*
```

Figure 2-21 (Part 1 of 3). MICR Document Processing Exit Routine Example 2

```
NONCTL   CLI   DIPKT,DIREJ               REJECTED DOCUMENT ?
         BNE   NORJCT                    N. RESET ERROR COUNTER
         LH    R10,RSCTR                 Y. INCREMENT
         LA    R10,1(R10)                ERROR
         STH   R10,RSCTR                 COUNTER
         CH    R10,MAXREJ                EXCESSIVE REJECTS ?
         BL    TABSRCH                   N. CHECK FOR DB/CR
         B     MESSAGE3                  PRODUCE ERROR MESSAGE
*
NORJCT   XC    RSCTR,RSCTR               CLEAR REJECT COUNTER
         B     TABSRCH                   CHECK FOR DB/CR
*
END      XC    RSA,RSA                   CLEAR THE WORK AREA
         B     RESUME                    RETURN WITH NO REQUEST
*
TABSRCH  MVI   DIFLD12+7,X'FF'           SET SIGN FOR UNPK INSTR
         MVO   DIFLD12+2(6),RSSER        SET AUX ONUS IN FL 12
         UNPK  DIFLD12+2(10),DIFLD12+2(6) CONV CHR DATA
         TR    DIFLD12+2(10),TRANSTBL    TRANS SPECL CHARS
*
         MVI   DIFLD14+138,X'FF'         SET SIGN FOR UNPK
         MVO   DIFLD14+131(8),RSACCT     SET ONUS IN FLD 14
         UNPK  DIFLD14+131(14),DIFLD14+131(8)  CONV DATA
         TR    DIFLD14+131(14),TRANSTBL  TRANS CHARS
*
         TM    DIFLAG2,DICRD             CREDIT ?
         BO    MOVECR                    Y. PROCESS CREDIT
         MVC   DIFLD15(8),DBID           N. SET DEBIT IDENTIFIER
         B     RESUME                    RETURN WITH NO REQUEST
*
MOVECR   MVC   RSSER(5),DIAUX            SAVE AUX ONUS
         MVC   RSACCT(7),DIONUS          SAVE ONUS
         MVC   DIFLD15(8),CRID           SET CREDIT ID
         MVI   DIFLD12+7,X'FF'           SET SIGN FOR UNPK
         MVO   DIFLD12+2(6),RSSER        SET NEW CR AUX ONUS
         UNPK  DIFLD12+2(10),DIFLD12+2(6) CONV CHAR DATA
         TR    DIFLD12+2(10),TRANSTBL    TRANS SPECL CHARS
*
         MVI   DIFLD14+138,X'FF'         SET SIGN FOR UNPK
         MVO   DIFLD14+131(8),RSACCT     SET NEW CR ONUS
         UNPK  DIFLD14+131(14),DIFLD14+131(8) CONV CHAR
         TR    DIFLD14+131(14),TRANSTBL  TRANS CHARS
         B     RESUME                    RETURN WITH NO REQUEST
*
MESSAGE3 MVC   M2DIGMSG,MSG3             SET ERROR MESSAGE
         MVI   M2RETCDE,M2STOPCP         SET 'STOP' RET CODE
         B     EXITPR                    EXIT ROUTINE
*
RESUME   MVI   M2RETCDE,M2REPCNT         SET 'RESUME' RET CODE
*
EXITPR   L     R13,4(,R13)               REST CALLER'S SAVE AREA @
         RETURN (14,12),,RC=(15)         RETURN
*
```

Figure 2-21 (Part 2 of 3). MICR Document Processing Exit Routine Example 2

```
TRANSTBL DC 240X'FF',C'0123456789 **-**'  TRANSLATE TBL
MAXREJ   DC    H'500'                     CONSECUTIVE REJECT LIMIT
BLKID    DC    CL8'BLOCK'                 BLOCK ID
BTHID    DC    CL8'BATCH'                 BATCH ID
DBID     DC    CL8'DEBIT'                 DEBIT ID
CRID     DC    CL8'CREDIT'                CREDIT ID
DUMMREC  DC  10X'FF'                      END OF STRNG DUMMY REC ID
MSG3     DC    CL40'EXCESSIVE REJECTS - CANCEL/RESTART'
*
         DKNREGS
*
PWRK     DSECT                           PRIVATE WORK AREA
SAVEAREA DS  18F                         OUR SAVE AREA
RSA      DS   0XL18                      AGGREGATE LENGTH ATTRIBUTE
RSCTR    DS    H                         REJECT COUNTER
RSINF    DS    H                         INF REJECT COUNTER
RSENDR   DS    H                         ENDORSE REJECT COUNTER
RSSER    DS    XL5                       AUXILIARY ON-US SAVE AREA
RSACCT   DS    XL7                       ON-US SAVE AREA
*
DIDSCT   DSECT                           MDS FORMATTED RECORD
         COPY  DIDSCT
*
         DKNMEX21                        MICR EXIT 2 INTRF CNTL BLK
*
         END
```

Figure 2-21 (Part 3 of 3). MICR Document Processing Exit Routine Example 2

MICR Exit 3 — Customize MICR Codeline Data Match Processing

This exit may be used to control the records sent to the document processor for codeline data matching (CDM). If active, the exit routine receives the records to be used for CDM before they are sent to the document processor. The exit routine can eliminate, change or add records to be used for CDM.

Only records corresponding to physical documents (control documents and items) and original reject records, delete/change High Performance Transaction System adjustment control records, and power/encode High Performance Transaction System adjustment control records, which are the records to be used for CDM by default, are passed to the exit routine. Section “Balancing and Power/Encoding Considerations” in the *CPCS-I Customization Guide* documents the High Performance Transaction System adjustment records and the power/encoding subsequent pass process.

On each call, the exit routine receives a buffer containing up to 16 contiguous records that have the same last ten digits in the sequence number field, which should be different versions of the same record. The exit routine must indicate, through a flag byte that precedes each record in the buffer, which records must be used for CDM. A flag byte value of X'00' indicates the record must be used for CDM. A flag byte value other than X'00' indicates the record must not be used for CDM. All records are passed to the exit routine with an initial flag value of X'00', with the exception of original reject records, change High Performance Transaction System adjustment control records, and power/encode High Performance Transaction System adjustment control records, which are passed with an initial flag value of X'FF'. The buffer start address (head pointer), buffer end address (end pointer), and first non-filled buffer slot address (tail pointer) are passed to the exit routine. The following figure shows a case in which the MICR task has placed three records in the buffer:

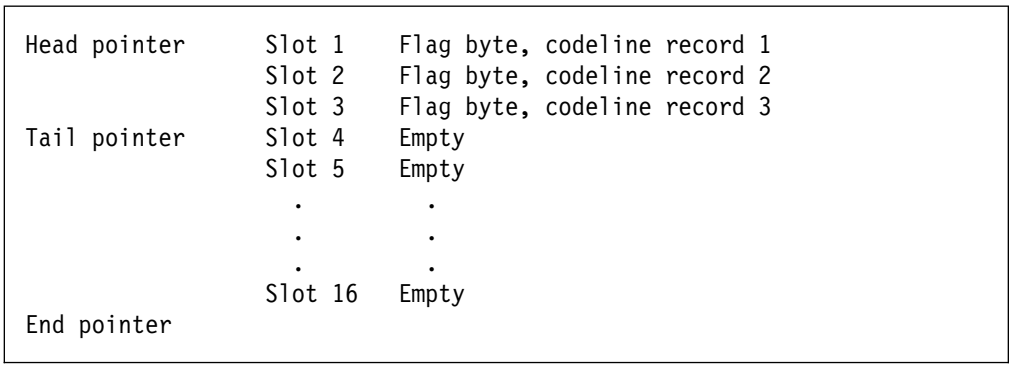

Figure 2-22. MICR CDM Exit Example Buffer

Eliminating CDM Records

Codeline records can be prevented from being used for CDM placing a value other than X'00' in their associated flag byte.

Changing CDM Records

Codeline records can have their fields modified by the exit routine.

See “Restrictions” on page 2-71.

Adding CDM Records

Codeline records can be inserted in the stream of records sent to the document processor by:

1. Formatting a codeline record in the slot to which the tail pointer points.
2. Placing a X'00' in the associated flag byte.
3. Moving the tail pointer by the length of the buffer slot.

See “Restrictions” on page 2-71.

Matching on Original Data and Power/Encoding from Corrected Data

The MICR exit 3 may be used on power/encoding passes to:

1. Power/encode rejected items from the corrected data, after having matched on the original data.
2. Power/encode High Performance Transaction System adjusted items from the changed data, after having matched on the original data.

To accomplish this, the MICR exit 3 routine must build the record to power/encode from into the work field of the record to match on, and flag as ‘for CDM’ only that record for that item sequence number. The document processor’s edit routine must also move the data to power/encode from into the process buffer on successful CDMs. The work field must be defined for the subsequent pass in the sort pattern definition (see “Sort Pattern Definition Record Formats - O Record” in the *CPCS-I Customization Guide*) for the field to be available on each record at MICR exit 3 routine execution time. The work field has a length equal to the total length of fields 1-8 plus 12 bytes (length of field 0), and it is passed to the sorter as the 9-15 field selected (this field must not be present in the MDS record). Also, if power/encoding from High Performance Transaction System balanced data, the MDIS exit (see “MDIS Exit — Modify M-string Distribution” on page 2-50) must allow the distribution of change and power/encode control records, which contain the original data to match on in the absence of associated original reject records.

Note: You can use DKNMXS31 as a template when you code MICR exit 3 user routines.

Activation

A different MICR CDM exit routine may be specified for each financial institution and sort type. The exit is active when one or more MICR CDM exit routine names are specified in the bank control file, or when a MICR CDM exit routine name is specified in a capture’s sort pattern definition, with the sort pattern definition taking precedence over the bank control file when both specify exit routine names (see “Bank Control File Record Formats” and “Sort-Pattern-Definition Record Formats” in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when the exit is activated/deactivated or when new exit routine versions are installed. The exit routine is dynamically loaded by the MICR BEGIN process and deleted by the MICR END process.

Linkage

The standard OS/390 linkage conventions are followed.

Information is passed in the following control block:

Table 2-26. MICR Exit 3 Communication Control Block

Control Block	Assembler Macro/ Copybook
Call Parameters	DKNMEX31

The private work area address field points to a 300 byte work area available to the user exit routine. The area is initialized to binary zeroes before the routine is entered for the first time (this also applies to capture restarts) in a capture. Its contents are preserved from call to call during the rest of the capture.

The Call parameters include the entry address and parameter list address of subroutine DKNMIPX. You need to call this subroutine to move and unformat compressed codeline records contained in the work field when you must send data placed in this area to the document processor, since the MICR task does not unformat the work field.

Copybooks DIDSCT, XF1MAP and RSCB may be used to map the compressed codeline record, and the XF1MAP and RSCB control blocks respectively.

Restrictions

MICR exit 3 routines:

- Must be written in assembler.
- Must be reentrant.

The MICR CDM user exit routine must be written in assembler.

Divider document records are passed to the exit routine, but never used for CDM (the exit routine flag byte settings are ignored).

The document processor does not return inserted records to the MICR task because they do not match within the document processor.

When you insert records, use caution to prevent the tail pointer from going beyond the end pointer. Changing storage beyond the end pointer can cause unpredictable results. The exit routine should not change any pointers other than the tail pointer.

Modifying a field used by the document processor to perform CDM might cause CDM on that document to fail. Changing field 8, the item-sequence number (DISEQ12), can cause missing and free item conditions in later CPCS-I processing.

Example 1

Operation:

Use the MICR CDM exit for the following:

1. Do not use records having a process control field value of X'000022' in the CDM process.
2. Insert a record after each record having a PC value of X'000011'. Added records have an ABA value of X'88888888' and the same amount as the last record that had a PC value of X'000033'. No insertion is performed if there are not slots available in the MICR CDM exit 3 buffer.

User Exit Routine:

```
MCDMEXT1 CSECT
MCDMEXT1 AMODE 24
MCDMEXT1 RMODE 24
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  MAP ROUTINE
*
         L     R2,0(R1)               GET INTERFACE CNTL BLK @
         USING M3INCNTL,R2            MAP INTERFACE CNTL BLK
         ICM   R3,B'1111',M3WAREA@    GET PRIV WRK AREA @
         USING PWRK,R3                MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SVE AREA @
         ST    R13,4(R14)             STORE BACKWRD SVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         L     R5,M3BHEAD@            GET BUFFER 1ST SLOT @
         USING BUFFER,R5              MAP BUFFER SLOT
*
LOOP     C     R5,M3TAIL@             CURR SLOT @ >= TAIL @?
         BNL   RETURN                 Y. ALL SLOTS PROCESSED
         CLC   DIPCTL,PC11            TYPE 11 ITEM ?
         BE    TYPE11                 Y. INSERT RECORD
         CLC   DIPCTL,PC22            TYPE 22 ITEM ?
         BE    TYPE22                 Y. DO NOT USE IT FOR CDM
         CLC   DIPCTL,PC33            TYPE 33 ITEM ?
         BNE   NEXT                   N. PROCESS NEXT SLOT
*
         MVC   SVAMT,DIAMT            SAVE ITEM AMOUNT
         B     NEXT                   PROCESS NEXT SLOT
*
TYPE22   MVI   FLAGBYTE,DISCARD       SET 'NOT FOR CDM' FLAG
*
NEXT     AH    R5,M3BSLOTL            GET NEXT BUFFER SLOT @
         B     LOOP                   PROCESS SLOT
*
```

Figure 2-23 (Part 1 of 2). MICR CDM Exit Routine Example 1

```
TYPE11   CLC   M3TAIL@,M3BEND@         TAIL @ >= END @ ?
         BNL   RETURN                  Y. NO ROOM FOR ADDITION
         L     R5,M3TAIL@              GET FIRST UNSUED SLOT @
*
         XC    DI,DI                   CLEAR INSERT RECORD
         MVC   DIABA,TKTID             SET TICKET ID
         MVC   DIAMT,SVAMT             SET AMT = PREV TYPE 56
         MVI   FLAGBYTE,USE            SET 'FOR CDM' FLAG
*
         AH    R5,M3BSLOTL             RESET TAIL @ TO
         ST    R5,M3TAIL@              NEXT BUFFER SLOT @
*
RETURN   XR    R15,R15                 CLEAR RETURN CODE
         L     R13,SAVEAREA+4          REST CALLER'S SVE AREA @
         XRETURN (14,12),,RC=(15)      EXIT ROUTINE
*
PC11     DC    X'000011'               RECORD TYPE 11
PC22     DC    X'000022'               RECORD TYPE 22
PC33     DC    X'000033'               RECORD TYPE 33
TKTID    DC    X'88888888'             TICKET ID
*
         DKNREGS                       REGISTER EQUATES
*
PWRK     DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       OUR SAVE AREA
SVAMT    DS    XL5                     TYPE 56 AMOUNT SAVE AREA
*
BUFFER   DSECT                         BUFFER SLOT
FLAGBYTE DS    X                       . CDM FLAG BYTE
USE      EQU   X'00'                   - USE RECORD FOR CDM
DISCARD  EQU   X'FF'                   - DO NOT USE FOR CDM
         COPY  DIDSCT                  . CODE-LINE DATA RECORD
WORKFLD  EQU    *                      . START OF WORK FIELD
*
         DKNMEX31                      MICR EXIT 3 INT CNTL BLK
*
         END
```

Figure 2-23 (Part 2 of 2). MICR CDM Exit Routine Example 1

Example 2

Operation:

Allows CDM on original data and power/encoding (P/E) from changed data on P/E subsequent passes.

The operation is performed looping twice through the CDM exit buffer.

On the first pass:

1. The changed record (or original record if it was never changed) is located.
2. Switches are set indicating if original reject, change control or P/E control records are found.

On the second pass:

1. The located changed record is moved into the work field of all the records in the buffer, calling DKNMIPX.
2. If any original reject, change control or P/E control record exists, the changed record is flagged as ‘not for CDM’.
3. If an original reject record exists, it is flagged as ‘for CDM’. Otherwise, if there is a change control or P/E control record, it is flagged as ‘for CDM’.

User Exit Routine:

```
MCDMEXT2 CSECT
MCDMEXT2 AMODE 24
MCDMEXT2 RMODE 24
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  MAP ROUTINE
*
         L     R2,0(R1)               GET INTERFACE CNTL BLK @
         USING M3INCNTL,R2            MAP INTERFACE CNTL BLK
         ICM   R3,B'1111',M3WAREA@    GET PRIV WRK AREA @
         USING PWRK,R3                MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SVE AREA @
         ST    R13,4(R14)             STORE BACKWRD SVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         L     R5,M3BHEAD@            GET BUFFER 1ST SLOT @
         USING BUFFER,R5              MAP BUFFER SLOT
*
         MVI   TYPESW,X'00'           INIT 'ONLY GOOD REC' SW
*
LOOP1    C     R5,M3TAIL@             CURR SLOT @ >= TAIL @ ?
         BNL   SETLOOP2               Y. BUFFER PROCESSED
         TM    DIFLAG2,DIMTCR         CONTROL RECORD ?
         BZ    GOODREC1               N. ITS IS THE GOOD REC
         CLI   DITYPEI,DIOREJ         ORIGINAL REJECT RECORD ?
         BE    SETSW11                Y. SET TYPE SWITCHOR CDM
         CLI   DITYPEI,DIBALDCH       DELETE/CHANGE CNTL REC?
         BE    SETSW12                Y. SET TYPE SWITCH
         CLI   DITYPEI,DIBALPE P/E    CONTROL RECORD ?
         BE    SETSW13                Y. SET TYPE SWITCH
*
```

Figure 2-24 (Part 1 of 3). MICR CDM Exit Routine Example 2

```
GOODREC1 LR    R3,R2                  SET GOOD RECORD @
         LA    R3,1(R3)               SET GOOD RECORD DI @
         B     NEXT1                  SLOT PROCESSED
*
SETSW11  MVI   TYPESW,X'01'           SET 'ORIGINAL REJ FOUND'
         B     NEXT1                  SLOT PROCESSED
*
SETSW12  TM    DI3BYTE2,DI3CHG        CHANGE CONTROL RECORD ?
         BZ    NEXT1                  N. DELETION
*
SETSW13  MVI   TYPESW,X'02'           SET 'CHANGE/P/E FOUND'
*
NEXT1    AH    R2,M3BSLOTL            POINT TO NEXT SLOT
         B     LOOP1                  PROCESS BUFFER NEXT SLOT
*
SETLOOP2 L     R2,M3BHEAD@            POINT TO BUFFER 1ST SLOT
*
LOOP2    C     R2,M3TAIL@             CURR SLOT @ >= TAIL @ ?
         BNL   RETURN                 Y. BUFFER PROCESSED
         CLC   M3CDMWKL,=H'0'         WORK FIELD PRESENT ?
         BE    NEXT2                  N. NOTHING TO DO
         ICM   R15,B'1111',M3MIPX@    GET DKNMIPX @
         STCM  R3,B'1111',M3CDELN@    SET GOOD REC AS SRC
         LA    R4,WORKFLD             SET WORK FIELD
         STCM  R4,B'1111',M3STREC@    AS TARGET
         LA    R1,M3MIPXPR            GET DKNMIPX PARM LIST @
         BASSM R14,R15                CALL DKNMIPX
         TM    DIFLAG2,DIMTCR         CONTROL RECORD ?
         BZ    GOODREC2               N. IT IS THE GOOD REC
         CLI   DITYPEI,DIOREJ         ORIGINAL REJECT REC ?
         BNE   CHECK21                N. CHECK FOR OTH TYPES
         MVI   FLAGBYTE,USE           SET 'FOR CDM' FLAG
         B     NEXT2                  SLOT PROCESSED
*
CHECK21  CLI   DITYPEI,DIBALDCH       DELETE/CHANGE CNTL REC?
         BNE   CHECK22                N. CHECK FOR P/E TYPE
         TM    DI3BYTE2,DI3CHG        CHANGE CONTROL RECORD ?
         BO    CHECK23                Y. SEEIF REJECT IN BUFF
*
CHECK22  CLI   DITYPEI,DIBALPE        P/E CONTROL RECORD ?
         BNE   GOODREC2               N. IT IS THE GOOD RECORD
*
CHECK23  CLI   TYPESW,X'01'           ORIGINAL REJECT IN BUFF?
         BE    NEXT2                  Y. SLOT PROCESSED
         MVI   FLAGBYTE,USE           SET 'NOT FOR CDM' FLAG
         B     NEXT2                  SLOT PROCESSED
*
GOODREC2 CLI   TYPESW,X'00'           ANY OR REJ, CHG OR P/E ?
         BE    NEXT2                  N. SLOT PROCESSED
         MVI   FLAGBYTE,DISCARD       SET 'NOT FOR CDM' FLAG
         B     NEXT2                  SLOT PROCESSED
*
NEXT2    AH    R2,M3SLOTL             POINT TO NEXT SLOT
         B     LOOP2                  PROCESS BUFFER NEXT SLOT
*
```

Figure 2-24 (Part 2 of 3). MICR CDM Exit Routine Example 2

```
RETURN   XR    R15,R15                 CLEAR RETURN CODE
         L     R13,SAVEAREA+4          REST CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      EXIT ROUTINE
*
         DKNREGS                       REGISTER EQUATES
*
PWRK     DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       OUR SAVE AREA
SVAMT    DS    XL5                     TYPE 56 AMOUNT SAVE AREA
TYPESW   DS    X                       TYPE SWITCH
*
BUFFER   DSECT                         BUFFER SLOT
FLAGBYTE DS    X                       . CDM FLAG BYTE
USE      EQU   X'00'                   - USE RECORD FOR CDM
DISCARD  EQU   X'FF'                   - DO NOT USE FOR CDM
         COPY  DIDSCT                  . CODE-LINE DATA RECORD
WORKFLD  EQU    *                      . START OF WORK FIELD
*
         DKNMEX31                      MICR EXIT 3 INT CNTL BLK
*
         END
```

Figure 2-24 (Part 3 of 3). MICR CDM Exit Routine Example 2

MICR Exit 4 — Customize Document Processor's Edit Routine and Table Load Processing (Non-XF Sort Types)

You can use this exit to dynamically build the names of the document processor's edit routine and tables, and modify their load process, on standard (non-expanded format) sort type captures.

Note: You can use the sample DKNMXS42 as a template when you code MICR exit 4 user routines.

Activation

The MICR exit 4 is a CPCS-I system-wide exit. The exit is active when its name is specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when the MICR exit 4 is activated or deactivated, or when a new exit routine version is installed. The exit routine is dynamically loaded by the MICR BEGIN process and deleted by the MICR END process.

Linkage

The standard OS/390 linkage conventions are followed.

Information is passed in the following control block:

Table 2-27. MICR Exit 4 Communication Control Block

Control Block	Assembler Macro/ Copybook
Call Parameters	DKNMEX41

The private work area address field points to a 300 byte work area available to the user exit routine. The area is initialized to binary zeros before the routine is entered for the first time (this also applies to capture restarts) in a capture. Its contents are preserved from call to call during the rest of the capture.

The diagnostic message must be returned along with "unsuccessful operation" return codes.

Restrictions

MICR exit 4 routines:

- Must be written in assembler.
- Must be reentrant.
- Are responsible for the entire edit routine and associated table load process (no edit routine/tables are loaded by MICR when this exit is active), which includes:
  1. Loading of the edit routine and tables.
  2. Desired validations of the loaded edit routine and tables (their prefix areas may be mapped with the SCI copybook).

3. Placement of the edit routines and tables in the 3890 core control block (mapped by the PROLOG macro when expanded with the ASSMTYP=M copybook).
4. Placement of the 3890 core edit routine prolog address in the SCB control block SETDVPGA field.
5. Deletion of the loaded edit routine and tables.

- Must perform module loads and deletes calling the MICR task load/delete module (DKNMLD).

  Module DKNMLD follows the standard MVS/ESA linkage conventions. A six fullword communication parameter list, containing the addresses of the SCB, RSCB, MTVT, module name, request code (X'01' - load, X'02' - delete) and returned entry point (load requests), is used. A completion code (0 - successful load/delete; 4 - failed load/delete; 8 - invalid request) is returned in register 15.

Example 1

Operation:

Use the MICR exit 4 to load edit routine EDRTNXXX and table TBXXXYYY, where XXX is the bank number and YYY is the sort type, performing the following validations:

1. The edit routine load address must be the prolog load address.
2. The table must be loaded outside the prolog.
3. The edit routine and table must not exceed the edit routine size limit.

User Exit Routine:

```
MEBLEXT4 CSECT
MEBLEXT4 AMODE 24
MEBLEXT4 RMODE 24
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  MAP ROUTINE
*
         L     R2,0(R1)               GET INTERFACE CNTL BLK @
         USING M4INCNTL,R2            MAP INTERFACE CNTL BLK
         ICM   R3,B'1111',M4WAREA@    GET PRIV WRK AREA @
         USING PWRK,R3                MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SVE AREA @
         ST    R13,4(R14)             STORE BACKWRD SVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
```

Figure 2-25 (Part 1 of 5). MICR Exit 4 Routine Example 1

```
         ICM   R8,B'1111',M4SCB@       GET SCB @
         USING SCBDSCT,R8              MAP SCB
         ICM   R9,B'1111',M4RSCB@      GET RSCB @
         USING RSCBDSCT,R9             MAP RSCB
         ICM   R10,B'1111',M4MTVT@     GET MTVT @
         USING MTVTDSCT,R10            MAP MTVT
*
LOADRTN  MVC   RTNPREF,RTNPR           BLD EDIT RTN NAME - PREFX
         MVC   RTNBBB,RSCBBANK                           - BANK
         ST    R8,SCB@                 SCB @ TO MLD PARM LIST
         ST    R9,RSCB@                RSCB @ TO MLD PARM LIST
         ST    R10,MTVT@               MTVT @ TO MLD PARM LIST
         LA    R5,RTNNAME              EDIT ROUTINE NAME @
         ST    R5,MODNAME@             TO MLD PARM LIST
         MVI   RCODE,LOAD              SET 'LOAD MODULE' REQUEST
         LA    R5,RCODE                REQUEST CODE @
         ST    R5,REQCDE@              TO MLD PARM LIST
         LA    R5,ENTRYPN@             MODULE ENTRY @ FLD @
         ST    R5,ENTPNT@              TO MLD PARM LIST
         LA    R1,MLDLIST              GET MLD PARM LIST @
         L     R15,MLD@                GET MLD @
         BASSM R14,R15                 LOAD EDIT ROUTINE
         LTR   R15,R15                 SUCCESSFUL COMPLETION ?
         BZ    VALRTN                  Y. VALIDATE LDED EDIT RTN
         MVC   M4DIGMSG,MSG1           SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),RTNNAME  MODULE NME TO MSG
         MVI   M4RETCDE,FAILED         SET 'REQ FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
VALRTN   L     R7,ENTRYPN@             GET LOADED RTN ENTRY @
         SH    R7,=H'8'                ADJUST @ TO RTN LOAD @
         USING SCIDSCT,R7              MAP LOADED ROUTINE
         L     R6,RSCBCUC              GET CUC @
         USING CUCDSCT,R6              MAP CUC
         CLC   SCILPA,=A(CUCPROLP)     EDIT RTN @ = PROL @?
         BE    VALRTN2                 Y. CONTINUE
         MVC   M4DIGMSG,MSG2           SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),RTNNAME  MODULE NME TO MSG
         MVI   M4RETCDE,FAILED         SET 'REQ FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
VALRTN2  LA    R2,CUCPRORG             GET CUC PROLOG @
         ST    R2,SETDVPGA             CUC PRLG@ TO SETDEV PARMS
         L     R3,SETDVPGL             GET MAX EDIT RTN LENGTH
         TM    SCBTYPE,SCBTYPXP        XP SORTER ?
         BO    VALRTN4                 Y. CONTINUE
         CLI   RSCBPPH,RSCBPPH0        PRIME PASS ?
         BE    VALRTN4                 Y. CONTINUE
         S     R3,CDMBUFSZ             SUBTRACT CDM BUFFERS SZE
*
VALRTN4  C     R3,SCILEN               EDIT RTN LEN > MAX LEN ?
         BNL   MOVERTN                 N. CONTINUE
         MVC   M4DIGMSG,MSG3           SET EROR MESSAGE
         MVC   M4DIGMSG+11(8),RTNNAME  MODULE NME TO MSG
         MVI   M4RETCDE,FAILED         SET 'REQ FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
```

Figure 2-25 (Part 2 of 5). MICR Exit 4 Routine Example 1

```
MOVERTN  LA    R4,SCIORG              GET LOADED EDIT RTN @
         L     R5,SCILEN              GET EDIT RTN LENGTH
         ICM   R5,8,=X'00'            SET PADDING CHAR
         MVCL  R2,R4                  MOVE EDIT ROUTINE TO CUC
*
DELRTN   MVI   RCODE,DELETE           SET 'DELETE MODULE' REQ
         LA    R1,MLDLIST             GET MLD PARM LIST @
         L     R15,MLD@               GET MLD @
         BASSM R14,R15                DELETE EDIT ROUTINE
         LTR   R15,R15                SUCCESSFUL COMPLETION ?
         BZ    LOADTBL                Y. CONTINUE
         MVC   M4DIGMSG,MSG4          SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),RTNNAME MODULE NME TO MSG
         MVI   M4RETCDE,FAILED        SET 'REQ FAILED' RET CDE
         B     RETURN                 EXIT ROUTINE
*
LOADTBL  XR    R5,R5                  CLEAR WORK REGISTER
         IC    R5,RSCBTYPE            CONVERT SORT TYPE
         CVD   R5,PSTYPE              TO DECIMAL
         UNPK  STYPE,PSTYPE+6(2)      CONVERT SORT TYPE
         OI    STYPE+2,NUMERIC        TO CHARACTER
         MVC   TBLPREF,TBLPR          BLD TABLE NAME - PREFIX
         MVC   TBLBBB,RSCBBANK                       - BANK
         MVC   TBLSSS,STYPE                          - SRT TYP
         LA    R5,TBLNAME             TABLE NAME @
         ST    R5,MODNAME@            TO MLD PARM LIST
         MVI   RCODE,LOAD             SET 'LOAD MODULE' REQ CDE
         LA    R1,MLDLIST             GET MLD PARM LIST @
         L     R15,MLD@               GET MLD @
         BASSM R14,R15                LOAD TABLE
         LTR   R15,R15                SUCCESSFUL COMPLETION ?
         BZ    VALTBL                 Y. CONTINUE
         MVC   M4DIGMSG,MSG1          SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),TBLNAME MODULE NME TO MSG
         MVI   M4RETCDE,FAILED        SET 'REQ FAILED' RET CDE
         B     RETURN                 EXIT ROUTINE
*
VALTBL   L     R7,ENTRYPN@            GET LOADED TABLE ENTRY @
         SH    R7,=H'8'               ADJUST @ TO TABLE LOAD @
         CLC   SCILPA,=A(CUCUSSLP)    TBL @ OUTSIDE PRLG ?
         BNL   VALTBL2                Y. CONTINUE
         MVC   M4DIGMSG,MSG5          SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),TBLNAME MODULE NME TO MSG
         MVI   M4RETCDE,FAILED        SET 'REQ FAILED' RET CDE
         B     RETURN                 EXIT ROUTINE
*
VALTBL2  L     R3,SETDVPGL            GET MAX EDIT RTN LENGTH
         LA    R3,208(R3)             MAX TABLE LENGTH = MAX
         S     R3,SCILPA              EDIT RTN LEN - TABLE @
         TM    SCBTYPE,SCBTYPXP       XP SORTER ?
         BO    VALTBL4                Y. CONTINUE
         CLI   RSCBPPH,RSCBPPH0       PRIME PASS ?
         BE    VALTBL4                Y. CONTINUE
         S     R3,CDMBUFSZ            SUBTRACT CDM BUFFERS SIZE
*
```

Figure 2-25 (Part 3 of 5). MICR Exit 4 Routine Example 1

```
VALTBL4  C     R3,SCILEN               TABLE LENGTH > MAX LEN ?
         BNL   MOVETBL                 N. CONTINUE
         MVC   M4DIGMSG,MSG3           SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),TBLNAME  MODULE NME TO MSG
         MVI   M4RETCDE,FAILED         SET 'REQU FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
MOVETBL  LA    R2,CUCORG               GET EDIT RTN AREA @
         A     R2,SCILPA               ADD TABLE OFFSET
         L     R4,SCIORG               GET LOADED TABLE @
         L     R5,SCILEN               GET TABLE LENGTH
         ICM   R5,B'1000',HEX0         SET PADDING CHAR
         MVCL  R2,R4                   MOVE TABLE TO CUC
*
DELTBL   MVI   RCODE,DELETE            SET 'DELETE MODULE' REQ
         LA    R1,MLDLIST              GET MLD PARM LIST @
         L     R15,MLD@                GET MLD @
         BASSM R14,R15                 DELETE TABLE
         LTR   R15,R15                 SUCCESSFUL COMPLETION ?
         BZ    OPDONE                  Y. CONTINUE
         MVC   M4DIGMSG,MSG4           SET ERROR MESSAGE
         MVC   M4DIGMSG+11(8),RTNNAME  MODULE NME TO MSG
         MVI   M4RETCDE,FAILED         SET 'REQ FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
OPDONE   MVI   M4RETCDE,M4VALOP        SET 'SUCCSSFL COMPLETN'
*
RETURN   L     R13,4(,R13)             REST CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN TO CALLER
*
         DKNREGS
*
RTNPR    DC    C'EDRTN'                EDIT ROUTINE NAME PREFIX
TBLPR    DC    C'TB'                   TABLE NAME NAME PREFIX
MLD@     DC    V(DKNMLD)               DKNMLD @
MSG1     DC    CL80'MEBLEXT4 - XXXXXXXX LOAD FAILED'
MSG2     DC    CL80'MEBLEXT4 - XXXXXXXX INVALID'
MSG3     DC    CL80'MEBLEXT4 - XXXXXXXX TOO LARGE'
MSG4     DC    CL80'MEBLEXT4 - XXXXXXXX DELETE FAILED'
MSG5     DC    CL80'MEBLEXT4 - XXXXXXXX INSIDE PROLOG'
CDMBUFSZ DC    F'3072'                 CDM BUFFERS SIZE
HEX0     DC    X'00'                   HEX ZEROES
NUMERIC  EQU   X'F0'                   NUMERIC UNSIGNED
FAILED   EQU   X'08'                   UNSUCCESSFUL OPERATION
*
PWRK     DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       SAVE AREA
*
RTNNAME  DS   0CL8                     EDIT ROUTINE NAME
RTNPREF  DS    CL5                     . PREFIX
RTNBBB   DS    CL3                     . BANK
*
```

Figure 2-25 (Part 4 of 5). MICR Exit 4 Routine Example 1

```
TBLNAME  DS   0CL8                    TABLE NAME
TBLPREF  DS    CL2                    . PREFIX
TBLBBB   DS    CL3                    . BANK
TBLSSS   DS    CL3                    . SORT TYPE
*
PSTYPE   DS    D                      WORK FIELD
ENTRYPN@ DS    F                      LOADED MODULE ENTRY @
STYPE    DS    CL3                    CHAR FORMAT SORT TYPE
*
MLDLIST  DS   0F                      DKNMLD PARM LIST
SCB@     DS    F                      . SCB @
RSCB@    DS    F                      . RSCB @
MTVT@    DS    F                      . MTVT @
MODNAME@ DS    F                      . MODULE NAME
REQCDE@  DS    F                      . REQUEST CODE @
ENTPNT@  DS    F                      . ENTRY @ @
*
RCODE    DS    X                      MLD REQUEST CODE
LOAD     EQU   X'01'                  . LOAD MODULE
DELETE   EQU   X'02'                  . DELETE MODULE
*
SCBDSCT  DSECT                        SCB
         COPY SCB
*
RSCBDSCT DSECT                        RSCB
         COPY RSCB
*
MTVTDSCT DSECT                        MTVT
         COPY MTVT
*
PRLG     PROLOG ASSMTYP=M             PROLOG
*
SCIDSCT  DSECT                        PROLOG PREFIX AREA
         COPY SCI
*
         DKNMEX41                     MICR EXIT 4 INT CNTL BLK
*
         END
```

Figure 2-25 (Part 5 of 5). MICR Exit 4 Routine Example 1

MICR Exits 5/6 — Customize Document Processor's Edit Routine and Table Load Processing (XF Sort Types)

These exits may be used to dynamically build the names of the document processor's edit routine and tables, and modify their load process, on expanded format sort type captures. Exit 5 applies to 2-byte addressing edit routine and tables. Exit 6 applies to 4-byte addressing edit routine and tables.

Notes:

1. You can use DKNMXS51 and DKNMXS61 as templates when you code MICR exit 5/6 user routines.
2. MICR exits 5/6 routines have the same names as CSBU exits 1/2 routines. Also, MICR exits 5/6 and CSBU exits 1/2 are activated simultaneously.

Activation

The MICR exits 5 and 6 are CPCS-I system-wide exits. The exits are active when their names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when the MICR exits 5 and 6 are activated or deactivated, or when new exit routine versions are installed. The exit routines are dynamically loaded by the MICR BEGIN process and deleted by the MICR END process.

Linkage

The standard OS/390 linkage conventions are followed.

Information is passed in the following control block:

Table 2-28. MICR Exit 5/6 Communication Control Block

Control Block	Assembler Macro/ Copybook
Call Parameters	DKNMEX41

The private work area address field points to a 300-byte work area available to the user exit routine. The area is initialized to binary zeroes before the routine is entered for the first time (this also applies to capture restarts) in a capture. Its contents are preserved from call to call during the rest of the capture.

The diagnostic message must be returned along with "unsuccessful operation" return codes.

Restrictions

MICR exit 5 routines:

- Must be written in assembler.
- Must be reentrant.
- May modify the edit routine loaded by MICR in the 3890 core area control block (mapped by the PROLOGX macro, when expanded with the

TYPE=SCIX,ASSMTYP=M (where X is 2 or 4, depending on the addressing mode) parameters, and addressed by the XF1MAP control block XF1CIB@ field (the XF1MAP control block is pointed to by the RSCB control block RSCBXF@), and load any associated tables. MICR will load a table, if indicated in the sort pattern definition for the capture, once the exit routine execution has completed successfully.

- Are responsible for the following:
  1. Desired validations of any edit routine and tables loaded by the exit routine (their prefix areas may be mapped with the PFX copybook).
  2. Placement of any loaded edit routine and tables in the 3890 core control block.
  3. Deletion of any edit routine and tables loaded by the exit routine.
- Must perform module loads and deletes with the LOAD/DELETE macro.

Example 1

Operation:

Use the MICR exit 5 to load edit tables PCTBLXXX and RTTBLXXX, where XXX is the XF sort type. Verify that the tables get loaded inside the edit routine already loaded by MICR. The edit routine and tables have 2-byte addressing.

User Exit Routine:

```
MEBLEXT5 CSECT
MEBLEXT5 AMODE 24
MEBLEXT5 RMODE 24
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  MAP ROUTINE
*
         L     R2,0(R1)               GET INTERFACE CNTL BLK @
         USING M5INCNTL,R2            MAP INTERFACE CNTL BLK
         ICM   R3,B'1111',M5WAREA@    GET PRIV WRK AREA @
         USING PWRK,R3                MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWRD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         ICM   R8,B'1111',M5PRLOG@    GET PROLOGX PREF @
         USING PFXDSCT,R8             MAP PROLOGX PREFIX AREA
         ICM   R9,B'1111',M5RSCB@     GET RSCB @
         USING RSCBDSCT,R9            MAP RSCB
         ICM   R7,B'1111',M5SCB@      GET SCB @
         USING SCBDSCT,R7             MAP SCB
*
```

Figure 2-26 (Part 1 of 4). MICR Exit 5 Routine Example 1

```
         XR    R5,R5                   CLEAR WORK REGISTER
         IC    R5,RSCBTYPE             CONVERT SORT TYPE
         CVD   R5,PSTYPE               TO DECIMAL
         UNPK  STYPE,PSTYPE+6(2)       CONVERT SORT TYPE
         OI    STYPE+2,NUMERIC         TO CHARACTER
         MVC   TBLPREF,PCPREF          BUILD
         MVC   TBLSSS,STYPE            PC TABLE NAME
         XC    TBLPROC,TBLPROC         CLEAR LOADTBL RET CDE
         BAS   R14,LOADTBL             PROCESS PC TABLE
         CLI   TBLPROC,TBLOK           PC TABLE PROCESSED ?
         BNE   FAILED                  N. STOP PROCESSING
*
         MVC   TBLPREF,RTPREF          BUILD RT TABLE NAME
         XC    TBLPROC,TBLPROC         CLEAR LOADTBL RET CDE
         BAS   R14,LOADTBL             PROCESS RT TABLE
         CLI   TBLPROC,TBLOK           RT TABLE PROCESSED ?
         BNE   FAILED                  N. STOP PROCESSING
         MVI   M5RETCDE,M5VALOP        SET 'SUCCSSFL CMPLTN'
         B     RETURN                  EXIT ROUTINE
*
FAILED   L     R15,4                   SET 'REQ FAILED' RET CDE
         CLI   TBLPROC,LOADER          TABLE LOAD ERROR?
         BNE   FAILDVRF                N. TABLE VERIFICATION ERR
         MVC   M5DIGMSG,MSG1           SET ERROR MESSAGE
         MVC   M5DIGMSG+11(8),TBLNAME  MODULE NME TO MSG
         MVI   M5RETCDE,FAILRQ         SET 'REQ FAILED' RET CDE
         B     RETURN                  EXIT ROUTINE
*
FAILDVRF MVC   M5DIGMSG,MSG2           SET ERROR MESSAGE
         MVC   M5DIGMSG+11(8),TBLNAME  MODULE NME TO MSG
         MVI   M5RETCDE,FAILRQ         SET 'REQ FAILED' RET CDE
*
RETURN   L     R13,4(,R13)             REST CALLER'S SVE AREA @
         XRETURN (14,12),,RC=(15)      RETURN TO CALLER
*
LOADTBL  LA    R5,TBLNAME              GET TABLE NAME @
         LOAD  EPLOC=(R5),ERRET=LDERR  LOAD TABLE
         B     VALTBL                  CONTINUE
*
LDERR    MVI   TBLPROC,LOADER          SET 'TABLE LOAD ERR' FLAG
         B     LOADEXIT                EXIT ROUTINE
*
```

Figure 2-26 (Part 2 of 4). MICR Exit 5 Routine Example 1

```
VALTBL   ST    R8,SAVER1               SAVE EDIT ROUTINE PREFX @
         MVC   EDRTNLEN,PFXSCILL       SAVE EDIT ROUTINE LEN
         ST    R0,TBLE@                SAVE TABLE ENTRY @
         LA    R4,PFXEND-PFXDSCT       GET PREFIX AREA LEN
         SR    R0,R4                   TBL LOAD @ = TBL ENTRY @
         LR    R8,R0                   -TBL PREFIX LENGTH
         MVC   TBLLEN,PFXSCILL         SAVE TABLE LENGTH
         MVC   TBLDISP,PFXSCIL@        SAVE TABLE DISPLACEMENT
         L     R4,PFXSCIL@             GET TABLE DISPLACEMENT
         A     R4,PFXSCILL             COMPUTE TABLE END @
         L     R8,SAVER1               RESTORE EDIT RTN PREFIX @
         C     R4,EDRTNLEN             TABLE INSIDE EDIT RTN ?
         BNH   MOVETBL                 Y. CONTINUE
         MVI   TBLPROC,SIZEER          SET 'SIZE ERROR' FLAG
         B     LOADEXIT                CONTINUE
*
MOVETBL  LR    R2,R8                   GET EDIT ROUTINE PREFIX @
         A     R2,TBLDISP              BUMP TO TABLE @
         L     R4,TBLE@                GET LOADED TABLE ENTRY @
         L     R5,TBLLEN               GET TABLE
         LR    R3,R5                   LENGTH
         ICM   R5,B'1000',HEX0         SET PADDING CHARACTER
         MVCL  R2,R4                   MOVE TABLE TO CUC
*
DELTBL   LA    R5,TBLNAME              TABLE NAME ?
         DELETE EPLOC=(R5)             DELETE TABLE
         MVI   TBLPROC,TBLOK           SET 'TABLE PROCESSED' FLG
*
LOADEXIT BSM   0,R14                   RETURN TO CALLING POINT
*
         DKNREGS
*
PCPREF   DC    C'PCTBL'                PC TABLE NAME PREFIX
RTPREF   DC    C'RTTBL'                RT TABLE NAME PREFIX
MSG1     DC    CL80'MEBLEXT5 - XXXXXXXX LOAD FAILED'
MSG2     DC    CL80'MEBLEXT5 - XXXXXXXX EXCEEDS RTN LEN'
HEX0     DC    X'00'                   HEX ZEROES
NUMERIC  EQU   X'F0'                   NUMERIC UNSIGNED
FAILRQ   EQU   X'08'                   LOAD FAILURE RETURN CODE
*
PWRK     DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       SAVE AREA
*
TBLNAME  DS   0CL8                     TABLE NAME
TBLPREF  DS    CL5                     . PREFIX
TBLSSS   DS    CL3                     . SORT TYPE
*
PSTYPE   DS    D                       WORK FIELD
SAVER1   DS    F                       REGISTER SAVE FIELD
ENTRYPN@ DS    F                       LOADED MODULE ENTRY @
EDRTNLEN DS    F                       EDIT ROUTINE LENGTH
TBLDISP  DS    F                       TABLE DISPLACEMENT
TBLE@    DS    F                       TABLE ENTRY @
TBLLEN   DS    F                       TABLE LENGTH
STYPE    DS    CL3                     CHARACTER FORMAT SRT TYPE
*
```

Figure 2-26 (Part 3 of 4). MICR Exit 5 Routine Example 1

```
TBLPROC  DS    X                        TABLE PROCESS INDICATOR
TBLOK    EQU   X'00'                    . PROCESSED
LOADER   EQU   X'04'                    . LOAD ERROR
SIZEER   EQU   X'08'                    . INCORRECT SIZE
*
*
SCBDSCT  DSECT                          SCB
         COPY SCB
*
RSCBDSCT DSECT                          RSCB
         COPY RSCB
*
PRLG     PROLOGX TYPE=SCI2,ASSMTYP=M    PROLOGX
*
PFXDSCT  DSECT                          PROLOG PREFIX AREA
         COPY PFX
*
         DKNMEX51                       MICR EXIT 5 INT CNTL BLK
*
         END
```

Figure 2-26 (Part 4 of 4). MICR Exit 5 Routine Example 1

MOLRI Exit — Modify Host's Edit Routine and Table Load Process

The host's codeline edit routine load exit allows the modification of the load and delete functions performed by the host's interface. If active, the exit routine is called by DKNMOLRI whenever it gets an open/close request code. The exit routine can:

- Load/delete the host codeline edit routine (and table, if applicable).
- Request host codeline edit routine (and table, if applicable) standard loading and deletion.
- Request the task's termination.

Notes:

1. Member MOLREXIT of CPCSI.V01R01.SDKNSAM2 may be used as a reference when coding MOLRI exit routines.
2. MOLRI is not a CPCS-I task. DKNMOLRI is a subroutine called by CPCS-I applications to perform host edit routine codeline record validations (see "Interfacing with Host Codeline Edit Routines" on page 1-129).

Loading and Deleting Host Codeline Edit Routine and Table

The exit routine may load the host codeline edit routine and optional table when called during the MOLRI open process (flag MUPAOPEN on). If so, it must also place the host codeline edit routine entry point address in the MUPA control block MUPASSEP field and the table (if any) address in field MUPASTEP, and set the appropriate return code (see linkage conventions below).

The exit routine may delete the host codeline edit routine and optional table when called during the MOLRI close process (flag MUPACLOS on). If so, it must also set the appropriate return code.

Requesting Standard Loading and Deletion.

The exit routine may request the loading and deletion of the host codeline edit routine and optional table by DKNMOLRI, by setting the appropriate return code. If so, DKNMOLRI loads/deletes the routine and table whose names are contained in fields MUPASSN and MUPASTN respectively, upon return from the exit routine. These fields, originally containing the names specified in the string header record (DKNMOLRI overlays the last byte in field MUPASSN with the character 'R'), may be modified by the exit routine.

Requesting Task Termination

The exit routine may request the task's termination by setting the appropriate return code described below. In this case, DKNMOLRI returns to the calling module with a return code of +20 in register 15, after sending message MOLRI10010, which includes the exit routine name, to the scroll log (see *CPCS-I Messages and Codes* for more information). The exit routine may also place an additional error message, preceded by a halfword that contains the message length, in the message buffer pointed to by field MUPAMSGA, to be used by the application task (DKNOLRR sends this message to the operator and supervisor terminals). DKNMOLRI returns to the calling module with a return code of +24, without performing any functions, on all subsequent calls.

Activation

A different exit routine can be specified for each financial institution. The exit is active when one or more MOLRI exit routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

A different exit routine may also be activated by specific interface, specifying the exit routine name in the MOLRIEX parameter of the DKNBLDL entry for each task invoking the host codeline edit routine interface. The MOLRIEX parameter lets you specify different exit routines for multiple tasks. See “Describing an Application Task’s Environment” in the *CPCS-I Customization Guide* for more information on the MOLRIEX parameter. The names specified by the MOLRIEX parameters override the names specified in the bank control file.

No re-gens, CPCS-I restarts or CHAPs are required when MOLRI exits are activated/deactivated via the bank control file, or when new exit routine versions are installed. Member DKNBLDL must be re-assembled, and CPCS-I must be restarted, when the exit is activated/deactivated via the MOLRIEX parameter.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-29. MOLRI Exit Communication Control Blocks

Control Block	Assembler Macro/ Copybook
CPCS-I Parameter List	DKNPARM
MICR/MOLRI User Parameter Area	MUPA

The exit routine may return the following completion codes in register 15:

Table 2-30. MOLRI Exit Routine Return Codes

Code (dec)	Description
+00	Host codeline edit routine (and table) have been successfully loaded/deleted.
+04	Host codeline edit routine (and table) must be loaded/deleted by MOLRI.
+08	Same as return code +4.
+12	Task processing must end.
+16	Same as return code +12.
> +16	Same as return code +12.

Restrictions

MOLRI exit routines:

- Must be written in assembler.
- Must be reentrant and serially reusable.

Example 1

Operation:

Use the MOLRI exit to:

1. Load host codeline edit routine S000XXXR and table T0000XXX, where XXX is the string's sort type, on 'open' calls.
2. Let DKNMOLRI delete the host codeline edit routine and table on 'close' calls.
3. Return an error message and request the task's termination if step 1 fails.

User Exit Routine:

```
MOLREXT1 CSECT
MOLREXT1 AMODE 31
MOLREXT1 RMODE ANY
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE ROUTINE
*
         L     R3,4(R1)               GET MUPA @
         USING MUPADSCT,R3            MAP MUPA
*
         CLI   MUPAUSER,STORDONE      WORK AREA ALREADY ALLOCATED ?
         BE    CHSAREAS               Y. SKIP 'STORAGE OBTAIN'
         STORAGE OBTAIN,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         LTR   R15,R15                SUCCESSFUL AREA ALLOCATION ?
         BNZ   STRGERR                N. PROCESS ERROR
         MVI   MUPAUSER,STORDONE      SET WRKAREA ALLOCATED SWITCH
         USING WRK,R5                 MAP PRIVATE WORK AREA
*
CHSAREAS LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         TM    MUPAFUNC,MUPAOPEN      'OPEN' CALL ?
         BZ    TSTCLOSE               N. SHOULD BE A 'CLOSE' CALL
*
         L     R4,MUPAIMGA            GET CODELINE RECORD @
         USING DIDSCT,R4              MAP CODELINE RECORD
         TM    DIFLAG2,DIMTCR         CONTROL RECORD
```

Figure 2-27 (Part 1 of 3). MOLRI Exit Routine Example 1

```
         BZ    SHDRERR                 N. WRONG RECORD PASSED
         CLI   DITYPEI,DISTGHD         STRING HEADER RECORD ?
         BNE   SHDRERR                 N. WRONG RECORD PASSED
*
         SR    R6,R6                   CLEAR WORK REGISTER
         ICM   R6,B'0001',DISPDNUM     GET SORT TYPE
         CVD   R6,PKWK                 CONVERT SORT TYPE TO DECIMAL
         UNPK  SORTTYPE,PKWK+6(2)      UNPACK SORT TYPE
         OI    SORTTYPE+2,X'F0'        MAKE SIGN DISPLAYABLE
*
         MVC   MUPASSN(4),RTPREFFX     BUILD ROUTINE NAME - PREFIX
         MVC   MUPASSN+4(3),SORTTYPE                      - SORT TYPE
         MVC   MUPASSN+7(1),RTSUFFIX                      - SUFFIX
         LOAD  EPLOC=MUPASSN,ERRET=RTLDERR  LOAD HOST EDIT ROUTINE
         ST    R0,MUPASSEP             SAVE ENTRY POINT IN MUPA
*
         MVC   MUPASTN(5),TBPREFFX     BUILD TABLE - PREFIX
         MVC   MUPASTN+5(3),SORTTYPE              - SORT TYPE
         LOAD  EPLOC=MUPASTN,ERRET=TBLDERR  LOAD TABLE
         ST    R0,MUPASTEP             SAVE ENTRY POINT IN MUPA
         LA    R15,0                   SET RETN CODE = 'LOADING DONE'
         B     RETURN                  EXIT PROGRAM
*
TSTCLOSE TM    MUPAFUNC,MUPACLOS       'CLOSE' CALL ?
         BZ    FUNCERR                 N. INVALID CALL
         LA    R15,4                   REQUEST DELETIONS BY MOLRI
         MVI   MUPAUSER+1,FREEWRK      WORK AREA MUST BE FREED
         B     RETURN                  EXIT PROGRAM
*
STRGERR  LA    R7,STRGEMSG             GET ERROR MESSAGE @
         B     SETERR                  PROCESS ERROR AND EXIT PROG
*
SHDRERR  LA    R7,SHDREMSG             GET ERROR MESSAGE @
         B     DEALWRK                 PROCESS ERROR AND EXIT PROG
*
RTLDERR  MVC   RTLDEMSG+13(8),MUPASSN  EDIT ROUTN NAME TO MESSAGE
         LA    R7,RTLDEMSG             GET ERROR MESSAGE @
         B     DEALWRK                 PROCESS ERROR AND EXIT PROG
*
TBLDERR  DELETE EPLOC=MUPASSN          DELETE HOST EDIT ROUTINE
         MVC   TBLDEMSG+13(8),MUPASTN  TABLE NAME TO MESSAGE
         LA    R7,TBLDEMSG             GET ERROR MESSAGE @
         B     DEALWRK                 PROCESS ERROR AND EXIT PROG
*
FUNCERR  LA    R7,FUNCEMSG             GET ERROR MESSAGE @
*
DEALWRK  MVI   MUPAUSER+1,FREEWRK      WORK AREA MUST BE FREED
*
SETERR   L     R6,MUPAMSGA             GET MOLRI'S MESSAGE BUFFER @
         MVC   0(40,R6),0(R7)          MOVE ERROR MESSAGE TO BUFFER
         LA    R15,12                  REQUEST TASK TERMINATION
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         CLI   MUPAUSER+1,FREEWRK      MUST WORK AREA BE FREED ?
         BNE   EXITPR                  N. EXIT PROGRAM
         LR    R10,R15                 SAVE RETURN CODE
```

Figure 2-27 (Part 2 of 3). MOLRI Exit Routine Example 1

```
         STORAGE RELEASE,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         XC    MUPAUSER(1),MUPAUSER    RESET WRKAREA ALLOCATED SWITCH
         XC    MUPAUSER+1(1),MUPAUSER  RESET FREE WRKAREA SWITCH
         LR    R15,R10                 RESTORE RETURN CODE
*
EXITPR   XRETURN (14,12),,RC=(15)      RETURN TO MOLRI
*
         DKNREGS                       REGISTER EQUATES
*
STORDONE EQU   X'01'                   WORK AREA ALLOCATED INDICATOR
FREEWRK  EQU   X'01'                   FREE WORK AREA INDICATOR
*
RTPREFFX DC    C'S000'                 EDIT ROUTINE NAME PREFIX
RTSUFFIX DC    C'R'                    EDIT ROUTINE NAME SUFFIX
TBPREFFX DC    C'T0000'                TABLE NAME SUFFIX
*
STRGEMSG DC    H'40',CL40'MOLREXT1 - STORAGE ALLOCATION ERROR'
SHDREMSG DC    H'40',CL40'MOLREXT1 - NO STRING HEADER PASSED'
RTLDEMSG DC    H'40',CL40'MOLREXT1 - XXXXXXXX ROUTINE LOAD ERROR'
TBLDEMSG DC    H'40',CL40'MOLREXT1 - XXXXXXXX TABLE LOAD ERROR'
FUNCEMSG DC    H'40',CL40'MOLREXT1 - INVALID CALL'
*
WRK      DSECT                         PRIVATE WORK AREA
SAVEAREA DS  18F                       . OUR SAVE AREA
PKWK     DS    D                       . PACK WORK FIELD
SORTTYPE DS    CL3                     . SORT TYPE IN CHAR FORMAT
WRKSIZE  EQU   *-WRK                   LENGTH OF PRIVATE WORK AREA
*
MUPADSCT DSECT                         MICR/MOLRI USER PARM AREA
         COPY  MUPA
*
DIDSCT   DSECT                         COMPRESSED CODELINE RECORD
         COPY  DIDSCT
*
         END
```

Figure 2-27 (Part 3 of 3). MOLRI Exit Routine Example 1

MRGE Exit — Modify HSRR MRGE Processing

The MRGE exit allows the modification of the codeline record match status.

This exit is called by MRGE, during HSSR merge processing, after each attempt to match a codeline record from the HSRR I-string with one from the reject D-string.

Note: Member MRGX000 of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding COBOL MRGE exit routines.

Activation

A different MRGE exit routine may be specified for each financial institution. The exit is active when one or more MRGE exit routine names are specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

No re-gens, CPCS-I restarts or CHAPs are required when new MRGE exit routine versions are installed.

The passed codeline records can be inspected but not modified.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-31. MRGF Exit Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Match Indicator	DKNMRGC0	
HSRR MRGE Profile Parameters	DKNHCMC0	DKNHCMA0
Compressed HSRR I-string Codeline Record to be written to the output M-string if the match indicator is on.	DKNCRDI	DIDSCT
Compressed Reject D-string Codeline Record used in the match.	DKNCRDI	DIDSCT
Compressed HSRR I-string Codeline Record used in the match.	DKNCRDI	DIDSCT

Restrictions

The MRGE exit routine:

- Should preferably be written in COBOL, but may also be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the MRGE exit to treat unmatched HSRR I-string codeline records having a user code of E4 and a process control of 001000, 005000 or 006000, as matched codeline records.

User Exit Routine:

```
 ID DIVISION.
 PROGRAM-ID. MRGX000.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  USER-CODES.
     05  UC-MATCHED                PIC X(01) VALUE X'E4'.

 01  PROCESS-CONTROL               PIC X(06).
     88  PC-MATCHED                          VALUE '001000'
                                                   '005000'
                                                   '006000'.
/
 LINKAGE SECTION.

 COPY DKNMRGC0.

 COPY DKNHCMC0                     REPLACING ==:HCDM:==
                                   BY        ==MRGE==.

 COPY DKNCRDI                      REPLACING ==DI==
                                   BY        ==DI-PROC==.

 COPY DKNCRDI                      REPLACING ==DI==
                                   BY        DI-MAST.

 COPY DKNCRDI                      REPLACING ==DI==
                                   BY        DI-PRORIG.
/
 PROCEDURE DIVISION                USING     SW-MATCH-FOUND
                                             MRGE-REC
                                             DI-PROC
                                             DI-MAST
                                             DI-PRORIG.

     IF  SW-MATCH-FOUND-NO
         IF  DI-PROC-USER-FLAG             =  UC-MATCHED
             MOVE DI-PROC-PROCESS-CONTROL  TO PROCESS-CONTROL
             IF  PC-MATCHED
                 SET MI-CODELINE-MATCHED   TO TRUE.

     GOBACK.
```

Figure 2-28. MRGE Exit Routine Example 1

MTASK Exit 1 — Perform Additional CPCS-I Startup Processing

The MTASK exit 1 may be used to perform additional initializations at CPCS-I startup time.

Notes:

1. See “VSMGR Exit 1 — Perform CPCS-I Startup VSAM Data Set Processing” on page 2-107 for a description of how to perform CPCS-I startup initializations for VSAM data sets that have been defined as CPCS-I resources.
2. Member MTSKINTX of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding MTASK exit 1 routines.

Activation

The MTASK initialization exit routine is active when its name has been specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-32. MTASK Exit 1 Communication Control Blocks

Control Block	Assembler Macro/ Copybook
CPCS-I Parameter List	DKNPARM

Restrictions

MTASK exit 1 routines should be written in assembler.

Example 1

Operation:

Use the MTASK exit 1 to initialize a QSAM data set (ddname DDTST) to binary zeroes on CPCS-I cold starts. The data set’s record length is 80 bytes.

User Exit Routine:

```
MTSKEXT1 CSECT
MTSKEXT1 AMODE 24
MTSKEXT1 RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         L     R4,0(R1)               GET PARM LIST @
         USING PARMLST,R4             MAP PARM LIST
*
         CLI   ETSKSTRT,COLDSTRT      COLD START ?
         BNE   RETURN                 N. EXIT ROUTINE
*
         OPEN  (DCB1,(UPDAT))         OPEN DATA SET
*
NEXTREC  GET   DCB1                   READ RECORD
         XC    0(80,R1),0(R1)         INITIALIZE RECORD TO X'00'S
         PUTX  DCB1                   REWRITE RECORD
         B     NEXTREC                PROCESS NEXT RECORD
*
CLOSEDCB CLOSE DCB1                   CLOSE DATA SET
*
RETURN   LA    R15,0                  ALWAYS SET RETURN CODE TO 0
         L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
COLDSTRT EQU   X'01'                  COLD START INDICATOR
*
DCB1     DCB   DDNAME=DDTST,DSORG=PS,MACRF=(GL,PL),EODAD=CLOSEDCB
*
         COPY DKNPARM                 CPCS PARM LIST
*
         END
```

Figure 2-29. MTASK Exit 1 Routine Example 1

MTASK Exit 2 — Perform Additional CPCS-I Shutdown Processing

The MTASK exit 2 may be used to perform additional termination processing at CPCS-I shutdown time.

Notes:

1. See “VSMGR Exit 2 — Perform CPCS-I Shutdown VSAM Data Set Processing” on page 2-110 for a description of how to perform CPCS-I shutdown terminations for VSAM data sets that have been defined as CPCS-I resources.
2. Member MTSKTRMX of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding MTASK exit 2 routines.

Activation

The MTASK termination exit routine is active when its name has been specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-33. MTASK Exit 1 Communication Control Blocks

Control Block	Assembler Macro/ Copybook
CPCS-I Parameter List	DKNPARM

Restrictions

MTASK exit 2 routines should be written in assembler.

Example 1

Operation:

Use the MTASK exit 2 to issue a WTO message, with the 20 character text pointed to by the CPCS-I PARM LIST user field PRMUS01, on CPCS shutdowns.

User Exit Routine:

```
MTSKEXT2 CSECT
MTSKEXT2 AMODE 24
MTSKEXT2 RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         L     R4,0(R1)               GET PARM LIST @
         USING PARMLST,R4             MAP PARM LIST
*
         L     R5,PRMUS01             GET MESSAGE TEXT @
         MVC   WTOLST+4(20),0(R5)     PUT MESSAGE TEXT IN WTO LIST
         LA    R1,WTOLST              GET WTO LIST @
         WTO   MF=(E,(1))             SEND MESSAGE
*
RETURN   LA    R15,0                  ALWAYS SET RETURN CODE TO 0
         L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
WTOLST   WTO   '                   ',MF=L  LIST FORM WTO
*
         COPY DKNPARM                 CPCS PARM LIST
*
         END
```

Figure 2-30. MTASK Exit 2 Routine Example 1

ODCR Exit — Access Adjustment Descriptions

This exit allows the access of adjustment code descriptions for outwork DCV reconciliation reporting.

Notes:

1. Member DKNDADJX of CPCSI.V01R01.SDKNSAM2 may be used as a reference when coding COBOL ODCR exit routines.
2. The same user routine used for the ODCR exit, is also used for the IDCR exit. Also, the ODCR exit and the IDCR exit are activated simultaneously.
3. See the IDCR exit section for information on activation, restrictions and linkage.

RCVY Exit — Modify Recovery Processing

The recovery exit allows to inspect/modify each recovered record, in both full (all strings) and selective string recovery modes.

The exit is called after each record conversion, before the compressed recovered codeline record is written.

Note: Member DKNLOGU of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding the recovery exit routine. It is also the default recovery exit routine name specified in the sample logging options member DKNRCVGN.

Activation

The recovery exit routine is active when the logging options member has been installed with an exit routine name specified in the RGENDEF macro USREXIT parameter (see “Logging CPCS-I Data” in the *CPCS-I Customization Guide* for information on how to define the logging options).

CPCS-I must be restarted whenever the recovery exit is activated or deactivated. A restart is not needed when a new exit routine version is installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-34. RCVY Exit Communication Control Blocks

Control Block	Assembler Macro/ Copybook
Application Task Control Block	DKNAPTCB
Reserved Area	
Compressed Recovered Codeline Record • Only this area can be modified.	DIDSCT
Logging/Recovery Characteristics	RDXREC
Codeline Record Characteristics	MDXREC

Restrictions

The recovery exit routine must be coded in assembler.

Example 1

Operation:

Use the recovery exit to modify recovered records having a user code of C8 and a blank process code, copying the last three digits of the account number into the process code.

User Exit Routine:

```
RCVYEXT1 CSECT
RCVYEXT1 AMODE 31
RCVYEXT1 RMODE ANY
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE ROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         LM    R3,R7,0(R1)            GET CONTROL BLOCK @'S
         USING DIDSCT,R5              MAP RECOVERED RECORD
*
         TM    DIFLAG2,DIMTCR         CONTROL RECORD ?
         BO    RETURN                 Y. EXIT PROGRAM
         CLC   DITYPEU,USERCD         SPECIAL ACCOUNT ?
         BNE   RETURN                 N. EXIT PROGRAM
         CLC   DIPCTL,BLANKS          SPLIT ACCOUNT ?
         BNE   RETURN                 N. EXIT PROGRAM
         MVC   DIPCTL+1(2),DIONUS+5   ON-US LAST 4 DIGITS TO PC
         MVO   DIPCTL+1(1),BLANKS(1)  BLANK TO PC 3RD DIGIT
*
RETURN   XR    R15,R15                CLEAR RETURN CODE
         L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN TO RCVY
*
         DKNREGS                      REGISTER EQUATES
*
USERCD   DC    X'C8'                  SPECIAL ACCOUNT USER CODE
BLANKS   DC   3X'AA'                  CPCS PACKED BLANKS
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
DIDSCT   DSECT                        COMPRESSED CODELINE RECORD
         COPY  DIDSCT
*
         END
```

Figure 2-31. RCVY Exit Routine Example 1

SECR Exit — Perform User Security Validations

This exit allows the installation of a non-RACF* CPCS-I security interface.

The exit is called at CPCS-I task attachment times and whenever tasks perform transaction level security checks.

Note: Member DKNSECRX of CPCSI.V01R01.SDKNSAM2 may be used as a template when coding user security exit routines.

Activation

The security exit routine is active when the CPCS-I MTASK task has been installed with the MDEF macro SCRTY=USER parameter and the exit routine name has been specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information).

CPCS-I must be restarted whenever the security exit is activated or deactivated, or a new exit routine version is installed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-35. SECR Exit Communication Control Blocks

Control Block	Assembler Macro/ Copybook
CPCS-I Parameter List	DKNPARM
CPCS-I Terminal Table Entry	DKNTENT
Transaction type: • Task-Name (issued at attachment time). • Transaction name. Some CPCS-I implemented transaction level security check IDs are: – INIT (initialization call; issued at CPCS-I startup time). – SIGNON (issued at SGON time; includes logon ID, old and new passwords). – SIGNOFF (issued at SGOF time). – IMAGESGN (High Performance Transaction System logon; includes logon ID, old and new passwords and operator ID) or ID).	SECRWKAR

Macro SECRPARM may be used to map the communication parameter list.

On attachment calls, the exit routine may return the following completion codes in register 15:

* Trademark of IBM

Table 2-36. SECR Exit Routine Attachment Return Codes

Code (dec)	CPCS-I Actions
+00	Task attached.
All others	Task not attached; Message DKNATASK32 issued.

On SIGNON transaction calls, the exit routine may return the following codes in register 15:

Table 2-37. SECR Exit Routine SIGNON Transaction Call Return Codes

Code (hex)	CPCS-I Actions
00000000	SGON allowed.
00000004	SGON not allowed; Message SGON-07 issued.
00000008	SGON not allowed; Message SGON-09 issued.
00000010	Same as code +00000004.
00000018	SGON not allowed; Message SGON-19 issued.
0000001C	SGON not allowed; Message SGON-14 issued.
00000104	Same as code 00000008.
00000108	SGON not allowed; Message SGON-10 issued.
0000010C	SGON not allowed; Message SGON-11 issued.
0000010D	SGON not allowed; Message SGON-18 issued.
00000110	SGON not allowed; Message SGON-12 issued.
00000114	Same as code 00000008.
00000118	SGON not allowed; Message SGON-13 issued.
0000011C	Same as code 0000001C.
00000120	Same as code 00000004.
00000124	Same as code 0000001C.
00000130	Same as code 00000008.
00000134	Same as code 00000008.
00000210	Same as code 00000004.
0000040C	Same as code 00000004.
00000410	Same as code 00000004.
00000414	Same as code 00000004.
00000418	Same as code 00000004.
00000808	Same as code 00000008.
00001006	SGON not allowed; Message SGON-21 issued.
00001007	SGON not allowed; Message SGON-22 issued.
All others	SGON not allowed; Message SGON-16 issued.

See the *CPCS-I Messages and Codes* for a description of these messages.

Note: The number of days remaining must be passed in bytes 16–31 of register 0, in printable format, when code 00000018 (password is going to expire) is returned.

Restrictions

User security exit routines:

- Should be written in assembler.
- Must be reentrant.

Example 1

Operation:

Use the user security exit routine to:

1. Allow CPCS-I access to users OP1, OP2 and OP3 only when they enter their respective passwords PW1, PW2 and PW3.
2. Allow the execution of task TASKA between 8:00AM and 4:00PM only.

User Exit Routine:

```
SECREXT1 CSECT
SECREXT1 AMODE 31
SECREXT1 RMODE ANY
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE ROUTINE
*
         L     R4,8(R1)               GET TRANSACTION AREA @
         USING TRANS,R4               MAP TRANSACTION AREA
*
         STORAGE OBTAIN,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         LTR   R15,R15                SUCCESSFUL AREA ALLOCATION ?
         BNZ   STRGERR                N. PROCESS ERROR
         USING WRK,R5                 MAP PRIVATE WORK AREA
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         CLC   TRANS(8),TRSIGNON      SIGNON CALL ?
         BNE   TSTTRAN                N. SEE IF TRANS LEVEL CHECK
*
         LA    R7,OPTENTS             GET OPER TABLE ENTRY COUNT
         LA    R8,OPTABLE             GET OPER TABLE ADDRESS
         USING OPENTRY,R8             MAP OPER TABLE ENTRIES
*
COMPOP   CLC   SWASOID,VALIDID        VALID OPER ID ?
         BNE   NEXTENT                N. BUMP TO NEXT ENTRY
         CLC   SWASOPW,VALIDPW        VALID PASSWORD ?
         BE    AUTH                   Y. OPER AUTHORIZED
*
```

Figure 2-32 (Part 1 of 3). SECR Exit Routine Example 1

```
NEXTENT  LA    R8,OPESIZE(R8)          GET NEXT OPER TABLE ENTRY @
         BCT   R7,COMPOP               COMPARE NEXT ENTRY, IF AVAIL
         B     NOTAUTH                 OPERATOR NOT AUTHORIZED
*
TSTTRAN  CLC   TRANS(5),TASKAID        TASKA SECURITY CHECK ?
         BNE   AUTH                    N. EXIT PROGRAM
         TIME  DEC                     GET CURRENT TIME
         ST    R0,PKTIME               STORE PACKED UNSIGNED TIME
         UNPK  UNPKHRS,PKTIME(2)       UNPACK HOURS
         CLC   UNPKHRS(2),HRSBOTLI     BEFORE AUTHORIZED TIME ?
         BL    NOTAUTH                 Y. TASK NOT AUTHORIZED
         CLC   UNPKHRS(2),HRSTOPLI     AFTER AUTHORIZED TIME ?
         BNH   AUTH                    N. TASK AUTHORIZED
*
NOTAUTH  L     R15,NOTALWD             SET 'NOT AUTHORIZED' RET CD
         B     RETURN                  EXIT PROGRAM
*
STRGERR  L     R15,MISCERR             SET 'MISC ERROR' RET CD
         B     RETURN                  EXIT PROGRAM
*
AUTH     L     R15,ALLOWED             SET 'AUTHORIZED' RET CD
*
RETURN   L     R13,SAVEAREA+4          RESTORE CALLER'S SAVE AREA @
         LR    R10,R15                 SAVE RETURN CODE
         STORAGE RELEASE,LENGTH=WRKSIZE,ADDR=(R5),SP=8,COND=YES
         LR    R15,R10                 RESTORE RETURN CODE
         XRETURN (14,12),,RC=(15)      RETURN TO SECR
*
         DKNREGS                       REGISTER EQUATES
*
TRSIGNON DC    CL8'SIGNON'             SIGNON TRANSACTION ID
TASKAID  DC    C'TASKA'                TASKA ID
*
HRSBOTLI DC    C'08'                   AUTHORIZATION TIME LOWER LIMIT
HRSTOPLI DC    C'15'                   AUTH TIME UPPER LIMIT - 1 MIN
*
         DS   0F                       RETURN CODES
MISCERR  DC    X'00000128'             . MISC SECURITY ERROR
NOTALWD  DC    X'00000104'             . NOT AUTHORIZED
ALLOWED  DC    X'00000000'             . AUTHORIZED
*
OPTABLE  DS   0H                       AUTHORIZED OPER TABLE
         DC    CL8'OP1',CL8'PW1'       . 1ST OPER ID & PASSWORD
         DC    CL8'OP2',CL8'PW2'       . 2ND OPER ID & PASSWORD
         DC    CL8'OP3',CL8'PW3'       . 3RD OPER ID & PASSWORD
OPTSIZE  EQU   *-OPTABLE               OPERATOR TABLE SIZE
*
OPENTRY  DSECT                         AUTHORIZED OPER TABLE ENTRY
VALIDID  DS    CL8                     . VALID OPER ID
VALIDPW  DS    CL8                     . VALID OPER PASSWORD
OPESIZE  EQU   *-OPENTRY               OPERATOR TABLE ENTRY SIZE
*
OPTENTS  EQU   OPTSIZE/OPESIZE         OPERATOR TABLE ENTRY COUNT
*
```

Figure 2-32 (Part 2 of 3). SECR Exit Routine Example 1

```
WRK      DSECT                          PRIVATE WORK AREA
SAVEAREA DS  18F                        . OUR SAVE AREA
PKTIME   DS    F                        . TIME CONVERSION WORK AREA
UNPKHRS  DS    XL3                      . HOURS    "       "    "
WRKSIZE  EQU   *-WRK                    LENGTH OF PRIVATE WORK AREA
*
CALLPRMS SECRPARM TYPE=DSECT            CALL PARAMETERS
*
TRANS    SECRWKAR TYPE=DSECT            TRANSACTION AREA
*
         END
```

Figure 2-32 (Part 3 of 3). SECR Exit Routine Example 1

VSMGR Exit 1 — Perform CPCS-I Startup VSAM Data Set Processing

The VSMGR exit 1 may be used to perform VSAM data set initializations at CPCS-I startup time. It may also request the termination of CPCS-I.

WTO message VSMGR30002, which includes the VSAM initialization exit routine name and its return code, is issued and CPCS-I is terminated, when a VSAM initialization exit routine returns a code other than zero.

Activation

A different initialization exit may be activated for each VSAM data set defined as a CPCS-I resource. The exit is activated by specifying the exit routine's name in the BEGEXIT parameter of its CPCS-I VSAM table entry.

Note: A VSAM data set is defined as a CPCS-I resource when it is listed in the CPCS-I VSAM table. See "Accessing the VSAM Table" in the *CPCS-I Customization Guide*, for information on how to define entries in the CPCS-I VSAM table.

CPCS-I must be restarted whenever a VSAM initialization exit is activated or deactivated.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-38. VSMGR Exit 1 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
CPCS-I VSAM Table Data Set Entry	DKNCRVSX	DKNVSENT
CPCS-I Parameter List	DKNCPARM	DKNPARM

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-39. VSMGR Exit 1 Routine Return Codes

Code (dec)	Request
+00	Continue CPCS-I startup processing.
All others	Terminate CPCS-I.

Restrictions

The accessed VSAM data sets must be defined as CPCS-I resources.

Example 1

Operation:

Use the VSMGR exit 1 to load a QSAM data set into a VSAM data set on CPCS-I cold starts. The VSAM data set initialization is performed invoking the IDCAMS services. DELETE, DEFINE and REPRO statements are contained in a data set allocated to ddname VSSYSIN1.

User Exit Routine:

```
VSMGEXT1 CSECT
VSMGEXT1 AMODE 24
VSMGEXT1 RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         L     R5,4(R1)               GET PARM LIST @
         USING PARMLST,R5             BASE PARM LIST
*
         CLC   ETSKSTRT,COLDSTRT      COLD START ?
         BE    INITDS                 Y. INITIALIZE DATA SET
         LA    R15,0                  SET 'RESUME START-UP' RET CD
         B     RETURN                 EXIT PROGRAM
*
INITDS   LA    R2,OPLIST              GET OPTIONS LIST @
         LA    R3,DDNLIST             GET DDNAME LIST @
         LA    R4,PAGENBR             GET PAGE NUMBER @
         LINK  EP=IDCAMS,PARAM=((R2),(R3),(R4)),VL=1,ERRET=LINKERR
         B     RETURN                 EXIT; <R15> = IDCAMS RET CD
*
LINKERR  LA    R15,32                 SET 'LINK FAILURE' RET CD
*
RETURN   L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
COLDSTRT DC    X'01'                  COLD START INDICATOR
*
OPLIST   DC    H'0'                   OPTIONS LIST
*
```

Figure 2-33 (Part 1 of 2). VSMGR Exit 1 Routine Example 1

```
DDNLIST  DS   0H                      ALTERNATE DDNAME LIST
         DC    H'48'                  . LIST LENGTH
         DC   4XL8'00'                . RESERVED
         DC    CL8'VSSYSIN1'          . ALTERNATE SYSIN DDNAME
         DC    CL8'SYSOUT'            . ALTERNATE SYSPRINT DDNAME
*
PAGENBR  DC    H'0'                   STARTING OUTPUT PAGE NUMBER
*
         COPY  DKNPARM                CPCS PARM LIST
*
         END
```

Figure 2-33 (Part 2 of 2). VSMGR Exit 1 Routine Example 1

VSMGR Exit 2 — Perform CPCS-I Shutdown VSAM Data Set Processing

The VSMGR exit 2 may be used to perform VSAM data set termination processing at CPCS-I shutdown time.

WTO message VSMGR30002, which includes the VSAM termination exit routine name and its return code, is issued when a VSAM termination exit routine returns a code other than zero.

Activation

A different termination exit may be activated for each VSAM data set defined as a CPCS-I resource. The exit is activated by specifying the exit routine's name in the ENDEXIT parameter of its CPCS-I VSAM table entry.

Note: A VSAM data set is defined as a CPCS-I resource when it is listed in the CPCS-I VSAM table. See "Accessing the VSAM Table" in the *CPCS-I Customization Guide*, for information on how to define entries in the CPCS-I VSAM table.

CPCS-I must be restarted whenever a VSAM termination exit is activated or deactivated.

A VSAM data set needs to be opened at least once, during CPCS-I processing, for its exit routine to be executed.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 2-40. VSMGR Exit 2 Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
CPCS-I VSAM Table Data Set Entry	DKNCRVSX	DKNVSENT
CPCS-I Parameter List	DKNCPARM	DKNPARM

The exit routine may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 2-41. VSMGR Exit 2 Routine Return Codes

Code (dec)	Request
+00	Continue CPCS-I shutdown processing.
All others	Issue WTO error message.

Restrictions

The accessed VSAM data sets must be defined as CPCS-I resources.

Example 1

Operation:

Use the VSMGR exit 2 to back up a VSAM data set into a QSAM data set on CPCS-I shutdowns. The VSAM data set is backed up invoking the IDCAMS services. The REPRO statement is contained in a data set allocated to ddname VSSYSIN2.

User Exit Routine:

```
VSMGEXT2 CSECT
VSMGEXT2 AMODE 24
VSMGEXT2 RMODE 24
*
         SAVE  (14,12)                SAVE REGISTERS
         BASR  R12,0                  GET ROUTINE @
         USING *,R12                  BASE SUBROUTINE
*
         LA    R14,SAVEAREA           POINT TO OUR SAVE AREA
         ST    R14,8(R13)             STORE FORWARD SAVE AREA @
         ST    R13,4(R14)             STORE BACKWARD SAVE AREA @
         LR    R13,R14                POINT TO NEW SAVE AREA
*
         LA    R2,OPLIST              GET OPTIONS LIST @
         LA    R3,DDNLIST             GET DDNAME LIST @
         LA    R4,PAGENBR             GET PAGE NUMBER @
         LINK  EP=IDCAMS,PARAM=((R2),(R3),(R4)),VL=1,ERRET=LINKERR
         B     RETURN                 EXIT; <R15> = IDCAMS RET CD
*
LINKERR  LA    R15,32                 SET 'LINK FAILURE' RET CD
*
RETURN   L     R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)     RETURN
*
         DKNREGS                      REGISTER EQUATES
*
SAVEAREA DS  18F                      OUR SAVE AREA
*
OPLIST   DC    H'0'                   OPTIONS LIST
*
DDNLIST  DS   0H                      ALTERNATE DDNAME LIST
         DC    H'48'                  . LIST LENGTH
         DC   4XL8'00'                . RESERVED
         DC    CL8'VSSYSIN2'          . ALTERNATE SYSIN DDNAME
         DC    CL8'SYSOUT'            . ALTERNATE SYSPRINT DDNAME
*
PAGENBR  DC    H'0'                   STARTING OUTPUT PAGE NUMBER
*
         END
```

Figure 2-34. VSMGR Exit 2 Routine Example 1

VTASK Exit 1 — VSCRL Special Buffer Processing

This exit may be used to perform special processing when the scroll buffer is processed with VSCRL.

Activation

The exit is active when its name is specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information.

No re-generations, CPCS-I restarts or CHAPs are required when a new VTASK exit 1 routine is installed.

Linkage

VSCRL is a section of VTASK. The VTASK task passes the following information to the VSCRL Special Scroll Buffer Processing user exit routine:

Table 2-42. VTASK Exit 1 Routine Entry Conventions

Register	Content
1	8-byte Parameter Address. The parameter contains: **Bytes 0–3** CPCS-I Parameter List Address **Bytes 4–7** Scroll Buffer Address

VTASK Exit 2 — VSCRL Special Scroll Initialization Processing

This exit may be used to perform special processing when the scroll is attached at initialization time.

Activation

The exit is active when its name is specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information.

No re-generations, CPCS-I restarts or CHAPs are required when a new VTASK exit 2 routine is installed.

Linkage

VSCRL is a section of VTASK. The VTASK task passes the following information to the VSCRL Special Scroll Initialization Processing user exit routine:

Table 2-43. VTASK Exit 2 Routine Entry Conventions

Register	Content
1	4-byte Parameter Address. The parameter contains: • Bytes 0–3 CPCS-I Parameter List Address

VTASK Exits 3/4 — VCOMS Special PCB Release Processing

These exits may be used to perform special processing when a Pool Control Block (PCB) is released with VCOMS.

Activation

The exits are active when their names are specified in the bank control file (see "Bank Control File Record Formats" in the *CPCS-I Customization Guide* for more information.

No re-generations, CPCS-I restarts or CHAPs are required when a new VTASK exit 3 or 4 routine is installed.

Linkage

VCOMS is a section of VTASK. The VTASK task passes the following information to the VCOMS Special PCB Release Processing user exit routine:

Table 2-44. VTASK Exit 3/4 Routine Entry Conventions

Register	Content
1	8-byte Parameter Address. The parameter contains: **Bytes 0–3** CPCS-I Parameter List Address **Bytes 4–7** Pool Control Block (PCB) Address

VTASK Exit 5 — VEXTS Special Terminal Logon Processing

This exit may be used to perform special processing when a terminal attempts to logon.

Activation

The exit is active when its name is specified in the bank control file (see “Bank Control File Record Formats” in the *CPCS-I Customization Guide* for more information.

No re-generations, CPCS-I restarts or CHAPs are required when a new VTASK exit 5 routine is installed.

Linkage

VEXTS is a section of VTASK.

The VTASK task passes the following information to the VEXTS Special Terminal Logon Processing user exit routine:

Table 2-45. VTASK Exit 5 Routine Entry Conventions

Register	Content
1	8-byte Parameter Address. The parameter contains: **Bytes 0–3** CPCS-I Parameter List Address **Bytes 4–7** Pool Control Block (PCB) Address

Chapter 3. Assembler Macros Provided by CPCS-I

Overview 3-3
ALDSN — Perform Dynamic Allocation Functions 3-4
ALLOC — Perform Dynamic Allocation Functions 3-10
CPCSNAME — Validate Name 3-16
DKNAMODE — Test/Set Addressing Mode 3-18
DKNFILL — Fill Segment with Padding Byte 3-21
DKNREGS — Generate Register Equates 3-23
DKNSBT — Search SCI Sort Binary Table 3-24
DKNXMODE — Test/Set Address Space Control Mode 3-26
FFLDL — Generate Field Length Equates/Constants 3-28
FPACK — Pack Unsigned a Zoned Numeric Field 3-31
FUNPK — Unpack a Packed Unsigned Field 3-34
XRETURN — Perform Program Exit Linkage 3-37

Overview

This chapter documents general-use CPCS-I macro instructions that may be included in CPCS-I assembler application programs and exit routines.

ALDSN — Perform Dynamic Allocation Functions

The ALDSN macro is used to dynamically allocate and unallocate data sets, and retrieve allocated data set information. No entries are required in the DKNDSAT table to perform these functions.

The dynamic allocation functions are performed issuing an SVC 99. See the MVS/ESA library for complete information on MVS/ESA dynamic allocation.

This macro sets the condition code based on the contents of register 15, which holds the SVC 99 return code.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede ALDSN.
ALDSN	
b	One or more blanks must follow ALDSN.
FUNC=ALLOC FUNC=UNALL FUNC=INFO	**Default:** FUNC=ALLOC.
,DDNAME=*address*	RX-type address.
,DSN=*address*	RX-type address.
,MEMBER=*address*	RX-type address.
,STATUS=*address*	RX-type address.
,NORMDSP=*address*	RX-type address.
,CONDDSP=*address*	RX-type address.
,SPACE=*address*	RX-type address.
,PRMSPAC=*address*	RX-type address.
,SECSPAC=*address*	RX-type address.
,DIRSPAC=*address*	RX-type address.
,RLSE=Y ,RLSE=*b*	**Default:** RLSE=*b*.
,ROUND=Y ,ROUND=*b*	**Default:** ROUND=*b*.
,VOLSER=*address*	RX-type address.
,VLSRLST=*address*	RX-type address.
,VOLCNT= *address*	RX-type address.
,UNIT=*address*	RX-type address.
,UNITCNT=*address*	RX-type address.

,SYSOUT=*address*	RX-type address.
,SYSPGNM=*address*	RX-type address.
,DEST=*address*	RX-type address.
,CLOSE=FREE ,CLOSE=*b*	**Default:** CLOSE=*b*.
,EXPDT=*address*	RX-type address.
,RETPD=*address*	RX-type address.
,DUMMY=Y ,DUMMY=*b*	**Default:** DUMMY=*b*.
,BLKSIZE=*address*	RX-type address.
,DEN=*address*	RX-type address.
,DSORG=*address*	RX-type address.
,KEYLEN=*number*	Decimal number.
,DCBREF=*address*	RX-type address.
,LIMCT=*address*	RX-type address.
,LRECL=*address*	RX-type address.
,OPTCD=*address*	RX-type address.
,RECFM=*address*	RX-type address.
,PASSWRD=*address*	RX-type address.
,OVSYSOT=*address*	RX-type address.
,RETDDN=*address*	RX-type address.
,RETDSN=*address*	RX-type address.
,RETMBR=*address*	RX-type address.
,RETSTAT=*address*	RX-type address.
,RETNDSP=*address*	RX-type address.
,RETCDSP=*address*	RX-type address.
,RETDSOR=*address*	RX-type address.
,RETVSER=*address*	RX-type address.
,ERROR=*address*	RX-type address.
,WORK=*address*	RX-type address.

Parameters

FUNC=ALLOC **FUNC=UNALL** **FUNC=INFO**	Specifies the dynamic allocation function being requested. FUNC=INFO is specified to request information about the current allocation environment, such as data set name, ddname, member name, data set organization, status, and normal and conditional disposition. FUNC=ALLOC is specified to request the dynamic allocation of a data set. FUNC=UNALL is specified to request the dynamic unallocation of a data set.
,DDNAME=*address*	Specifies an 8-byte field that contains the data definition name (ddname) involved in the dynamic allocation operation.
,DSN=*address*	Specifies a 44-byte field that contains the name of the data set involved in the dynamic allocation operation.
,MEMBER=*address*	Specifies an 8-byte field that contains the name of the member involved in the dynamic allocation operation.
,STATUS=*address*	Specifies a 1-byte field that contains the SVC-99 data set status code.
,NORMDSP=*address*	Specifies a 1-byte field that contains the SVC-99 data set normal disposition code.
CONDDSP=*address*	Specifies a 1-byte field that contains the SVC-99 data set conditional disposition code.
,SPACE=*address*	Specifies a 1-byte field that contains the SVC-99 data set space unit code.
,PRMSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the amount of primary DASD space requested on data set dynamic allocations. The SPACE parameter must also be specified when this parameter is used.
,SECSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the amount of secondary DASD space requested on data set dynamic allocations. The SPACE parameter must also be specified when this parameter is used.
,DIRSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the requested number of directory blocks on data set dynamic allocations.
,RLSE=Y **,RLSE=***b*	RLSE=Y specifies unused space deletion at data set closure.
,ROUND=Y **,ROUND=***b*	ROUND=Y specifies space allocation in whole cylinders.
,VOLSER=*address*	Specifies a 6-byte unpacked field that contains the volume serial number of the data set being allocated.
,VLSRLST=*address*	Specifies the data set's volume sequence number.
,VOLCNT=*address*	Specifies a 1-byte binary field that contains the number of volumes of the data set being allocated.
,UNIT=*address*	Specifies a 6-byte character field that contains the data set's unit group name.
,UNITCNT=*address*	Specifies a 1-byte binary field that contains the number of devices to be allocated.

,SYSOUT=*address*	Specifies a 1-byte character field that contains the class of the SYSOUT data set being allocated.
,SYSPGNM=*address*	Specifies a 8-byte character field that contains the SYSOUT program name.
,DEST=*address*	Specifies a 1-byte character field that contains the SYSOUT destination class.
,CLOSE=FREE **,CLOSE=***b*	`CLOSE=FREE` specifies data set freeing at closure time.
,EXPDT=*address*	Specifies a 5-byte character field that contains the data set's Julian (YYJJJ) expiration date.
,RETPD=*address*	Specifies a 2-byte field that contains the data set's retention period.
,DUMMY=Y **,DUMMY=***b*	`DUMMY=Y` specifies the allocation of a dummy data set.
,BLKSIZE=*address*	Specifies a 2-byte binary number that contains the data set's blocksize in bytes.
,DEN=*address*	Specifies a 1-byte field that contains the SVC 99 tape density code.
,DSORG=*address.*	Specifies a 1-byte field that contains the SVC 99 data set organization code.
,KEYLEN=*number*	Specifies a BDAM data set's key length.
,DCBREF=*address*	Specifies a 44-byte field that contains the name of the data set from which DCB information is to be retrieved.
,LIMCT=*address*	Specifies a 1-byte field containing the search limit.
,LRECL=*address*	Specifies a 2-byte field containing the data set's logical record length.
,OPTCD=*address*	Specifies a 1-byte field that contains the SVC 99 optional services code.
,RECFM=*address*	Specifies a 1-byte field that contains the SVC 99 data set record format code.
,PASSWRD=*address*	Specifies an 8-byte field that contains the data set's password.
,OVSYSOT=*address*	Specifies a 1-byte character field that contains an overriding SYSOUT class.
,RETDDN=*address*	Specifies an 8-byte field that contains the retrieved ddname.
,RETDSN=*address*	Specifies an 44-byte field that contains the retrieved data set name.
,RETMBR=*address*	Specifies an 8-byte field that contains the retrieved data member name.
,RETSTAT=*address*	Specifies an 1-byte field that contains the retrieved data set's status code.
,RETNDSP=*address*	Specifies an 1-byte field that contains the retrieved data set's normal disposition code.
,RETCDSP=*address*	Specifies an 1-byte field that contains the retrieved data set's conditional disposition code.
,RETDSOR=*address*	Specifies an 1-byte field that contains the retrieved data set's organization code.

,RETVSER=*address*	Specifies an 6-byte field that contains the retrieved data set's volume serial number.
,ERROR=*address*	Specifies a 4-byte field that holds the SVC 99 secondary return code.
,WORK=*address*	Specifies a 250-byte work area used to build the SVC 99 request block. This parameter is always required.

Example 1

Operation:

Dynamically create data set DATA.SET1, using UNIT=3390, VOL=SER= TSO001, SPACE=(TRK,(10,3)), DISP=(NEW,CATLG,DELETE), DCB=(LRECL=100, BLKSIZE=1000,RECFM=FB), allocating it to ddname DDNAME1.

Macro Definition:

```
         ALDSN WORK=WK1,DDNAME=DD1,DSN=DSN1,STATUS=ST1,NORMDSP=NDSP1,
               CONDDSP=CDSP1,SPACE=SPC1,PRMSPAC=PSPC1,SECSPAC=SSPC1,
               VOLSER=VSR1,UNIT=UNT1,BLKSIZE=BSZ1,DSORG=DSRG1,
               LRECL=LREC1,RECFM=RFMT1,ERROR=ER1
*
         BNZ TESTER1                FAILED ALLOCATION; TEST RET CDES
         ..............
         ..............
         ..............
*
WK1      DS  XL250                  REQUEST BLOCK WORK AREA
DD1      DC  CL8'DDNAME1'           DDNAME
DSN1     DC  CL44'DATA.SET1'        DSN
ST1      DC  XL1'04'                STATUS = NEW
NDSP1    DC  XL1'02'                NORMAL DISPOSITION = CATLG
CDSP1    DC  XL1'04'                CONDITIONAL DISPOSITION = DELETE
SPC1     DC  XL1'07'                SPACE UNITS = TRKS
PSPC1    DC  XL3'000010'            PRIMARY SPACE = 10 UNITS
SSPC1    DC  XL3'000003'            SECONDARY SPACE = 3 UNITS
VSR1     DC  CL6'TSO001'            VOL-SER = TSO001
UNT1     DC  CL8'3390'              UNIT = 3390
BSZ1     DC  XL2'03E8'              BLOCK SIZE = 1000
DSRG1    DC  XL1'40'                ORGANIZATION = PS
LREC1    DC  XL2'0064'              RECORD SIZE = 100
RFMT1    DC  XL1'90'                RECORD FORMAT = FB
ER1      DS  CL4                    RETURN CODE AREA
```

Figure 3-1. ALDSN Macro Definition - Example 1

Example 2

Operation:

Allocate ddname DDNAME1 to cataloged data set DATA.SET1 with a status disposition of SHR.

Macro Definition:

```
         ALLOC WORK=WK1,DDNAME=DD1,DSN=DSN1,STATUS=ST1,ERROR=ER1
*
         BNZ TESTER1                FAILED ALLOCATION; TEST RET CDES
         ..............
         ..............
         ..............
*
WK1      DS  XL250                  REQUEST BLOCK WORK AREA
DD1      DC  CL8'DDNAME1'           DDNAME
DSN1     DC  CL44'DATA.SET1'        DSN
ST1      DC  XL1'08'                STATUS = SHR
ER1      DS  CL4                    RETURN CODE AREA
```

Figure 3-2. ALDSN Macro Definition - Example 2

ALLOC — Perform Dynamic Allocation Functions

The ALLOC macro is used to dynamically allocate and unallocate data sets, and retrieve allocated data set information. No entries are required in the DKNDSAT table to perform these functions.

The dynamic allocation functions are performed issuing an SVC 99. See the MVS/ESA library for complete information on MVS/ESA dynamic allocation.

This macro sets the condition code based on the contents of register 15, which holds the SVC 99 return code.

This macro provides the same services as the ALDSN macro, with the following differences:

1. Some of its parameters need to be hardcoded.
2. The SVC 99 codes are set by the macro expansion.
3. It accepts the parameters DEFER and PRVT.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede ALLOC.
ALLOC	
b	One or more blanks must follow ALLOC.
FUNC=ALLOC FUNC=UNALL FUNC=INFO	**Default:** FUNC=ALLOC.
,DDNAME=*address*	RX-type address.
,DSN=*address*	RX-type address.
,MEMBER=*address*	RX-type address.
,STATUS=SHR ,STATUS=NEW ,STATUS=OLD ,STATUS=MOD	
,NORMDSP=CATLG ,NORMDSP=UNCATLG ,NORMDSP=DELETE ,NORMDSP=KEEP	
,CONDDSP=CATLG ,CONDDSP=UNCATLG ,CONDDSP=DELETE ,CONDDSP=KEEP	
,SPACE=TRK ,SPACE=CYL ,SPACE=BLK	
,PRMSPAC=*address*	RX-type address.

,SECSPAC=*address*	RX-type address.
,DIRSPAC=*address*	RX-type address.
,RLSE=Y	
,RLSE=*b*	**Default:** RLSE=*b*.
,ROUND=Y	
,ROUND=*b*	**Default:** ROUND=*b*.
,VOLSER=*address*	RX-type address.
,VLSRLST=*address*	RX-type address.
,VOLCNT= *address*	RX-type address.
,UNIT=*address*	RX-type address.
,PRVT=Y	
,PRVT=N	**Default:** PRVT=N.
,UNITCNT=*address*	RX-type address.
,SYSOUT=*address*	RX-type address.
,SYSPGNM=*address*	RX-type address.
,DEST=*address*	RX-type address.
,CLOSE=FREE	
,CLOSE=*b*	**Default:** CLOSE=*b*.
,EXPDT=*address*	RX-type address.
,RETPD=*address*	RX-type address.
,DUMMY=Y	
,DUMMY=*b*	**Default:** DUMMY=*b*.
,BLKSIZE=*address*	RX-type address.
,DEN=0	
,DEN=1	
,DEN=2	
,DEN=3	
,DEN=4	
,DEFER=Y	
,DEFER=N	**Default:** DEFER=N.
,DSORG=PS	
,DSORG=PO	
,DSORG=DA	
,KEYLEN=*number*	Decimal number.
,DCBREF=*address*	RX-type address.
,LIMCT=*address*	RX-type address.
,LRECL=*address*	RX-type address.
,OPTCD=E	
,OPTCD=R	

```
,RECFM=F
,RECFM=FA
,RECFM=FM
,RECFM=FB
,RECFM=FBA
,RECFM=FBM
,RECFM=V
,RECFM=VA
,RECFM=VM
,RECFM=VB
,RECFM=VBA
,RECFM=VBM

,PASSWRD=address          RX-type address.

,OVSYSOT=address          RX-type address.

,RETDDN=address           RX-type address.

,RETDSN=address           RX-type address.

,RETMBR=address           RX-type address.

,RETSTAT=address          RX-type address.

,RETNDSP=address          RX-type address.

,RETCDSP=address          RX-type address.

,RETDSOR=address          RX-type address.

,RETVSER=address          RX-type address.

,ERROR=address            RX-type address.

,WORK=address             RX-type address.
```

Parameters

FUNC=ALLOC
FUNC=UNALL
FUNC=INFO

Specifies the dynamic allocation function being requested.

`FUNC=INFO` is specified to request information about the current allocation environment, such as data set name, ddname, member name, data set organization, status, and normal and conditional disposition.

`FUNC=ALLOC` is specified to request the dynamic allocation of a data set.

`FUNC=UNALL` is specified to request the dynamic unallocation of a data set.

,DDNAME=*address*

Specifies an 8-byte field that contains the data definition name (ddname) involved in the dynamic allocation operation.

,DSN=*address*

Specifies a 44-byte field that contains the name of the data set involved in the dynamic allocation operation.

,MEMBER=*address*

Specifies an 8-byte field that contains the name of the member involved in the dynamic allocation operation.

,STATUS=SHR
,STATUS=NEW
,STATUS=OLD
,STATUS=MOD

Specifies the status of the data set involved in the dynamic allocation operation.

,NORMDSP=CATLG **,NORMDSP=UNCATLG** **,NORMDSP=DELETE** **,NORMDSP=KEEP**	Specifies the normal disposition of the data set involved in the dynamic allocation operation.
,CONDDSP=CATLG **,CONDDSP=UNCATLG** **,CONDDSP=DELETE** **,CONDDSP=KEEP**	Specifies the conditional disposition of the data set involved in the dynamic allocation operation.
,SPACE=TRK **,SPACE=CYL** **,SPACE=BLK**	Specifies the units in which DASD space is indicated in data set dynamic allocations.
,PRMSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the amount of primary DASD space requested on data set dynamic allocations. The SPACE parameter must also be specified when this parameter is used.
,SECSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the amount of secondary DASD space requested on data set dynamic allocations. The SPACE parameter must also be specified when this parameter is used.
,DIRSPAC=*address*	Specifies a 3-byte packed unsigned field that contains the requested number of directory blocks on data set dynamic allocations.
,RLSE=Y **,RLSE=***b*	`RLSE=Y` specifies unused space deletion at data set closure.
,ROUND=Y **,ROUND=***b*	`ROUND=Y` specifies space allocation in whole cylinders.
,VOLSER=*address*	Specifies a 6-byte unpacked field that contains the volume serial number of the data set being allocated.
,VLSRLST=*address*	Specifies the data set's volume sequence number.
,VOLCNT=*address*	Specifies a 1-byte binary field that contains the number of volumes of the data set being allocated.
,UNIT=*address*	Specifies a 6-byte character field that contains the data set's unit group name.
,PRVT=Y **,PRVT=N**	Specifies the private volume use attribute.
,UNITCNT=*address*	Specifies a 1-byte binary field that contains the number of devices to be allocated.
,SYSOUT=*address*	Specifies a 1-byte character field that contains the class of the SYSOUT data set being allocated.
,SYSPGNM=*address*	Specifies a 8-byte character field that contains the SYSOUT program name.
,DEST=*address*	Specifies a 1-byte character field that contains the SYSOUT destination class.
,CLOSE=FREE **,CLOSE=***b*	`CLOSE=FREE` specifies data set freeing at closure time.
,EXPDT=*address*	Specifies a 5-byte character field that contains the data set's Julian (YYJJJ) expiration date.
,RETPD=*address*	Specifies a 2-byte field that contains the data set's retention period.
,DUMMY=Y **,DUMMY=***b*	`DUMMY=Y` specifies the allocation of a dummy data set.

,BLKSIZE=*address*	Specifies a 2-byte binary number that contains the data set's blocksize in bytes.
,DEN=0 **,DEN=1** **,DEN=2** **,DEN=3** **,DEN=4**	Specifies tape density settings.
,DEFER=Y **,DEFER=N**	Specifies that the volume(s) mount should be deferred until the allocated data set is opened.
,DSORG=PS **,DSORG=PO** **,DSORG=DA**	Specifies the data set organization.
,KEYLEN=*number*	Specifies a BDAM data set's key length.
,DCBREF=*address*	Specifies a 44-byte field that contains the name of the data set from which DCB information is to be retrieved.
,LIMCT=*address*	Specifies a 1-byte field containing the search limit.
,LRECL=*address*	Specifies a 2-byte field containing the data set's logical record length.
,OPTCD=E **,OPTCD=R**	Specifies optional servicess.
,RECFM=F **,RECFM=FA** **,RECFM=FM** **,RECFM=FB** **,RECFM=FBA** **,RECFM=FBM** **,RECFM=V** **,RECFM=VA** **,RECFM=VM** **,RECFM=VB** **,RECFM=VBA** **,RECFM=VBM**	Specifies the data set's record format.
,PASSWRD=*address*	Specifies an 8-byte field that contains the data set's password.
,OVSYSOT=*address*	Specifies a 1-byte character field that contains an overriding SYSOUT class.
,RETDDN=*address*	Specifies an 8-byte field that contains the retrieved ddname.
,RETDSN=*address*	Specifies an 44-byte field that contains the retrieved data set name.
,RETMBR=*address*	Specifies an 8-byte field that contains the retrieved data member name.
,RETSTAT=*address*	Specifies an 1-byte field that contains the retrieved data set's status code.
,RETNDSP=*address*	Specifies an 1-byte field that contains the retrieved data set's normal disposition code.
,RETCDSP=*address*	Specifies an 1-byte field that contains the retrieved data set's conditional disposition code.
,RETDSOR=*address*	Specifies an 1-byte field that contains the retrieved data set's organization code.
,RETVSER=*address*	Specifies an 6-byte field that contains the retrieved data set's volume serial number.

,ERROR=*address* Specifies a 4-byte field that holds the SVC 99 secondary return code.

,WORK=*address* Specifies a 250-byte work area used to build the SVC 99 request block. This parameter is always required.

Example 1

Operation:

Allocate ddname DDNAME1 to cataloged data set DATA.SET1 with a status disposition of SHR.

Macro Definition:

```
         ALLOC WORK=WK1,DDNAME=DD1,DSN=DSN1,STATUS=SHR,ERROR=ER1
*
         BNZ TESTER1                FAILED ALLOCATION; TEST RET CDES
         .................
         .................
         .................
*
WK1      DS  XL250                  REQUEST BLOCK WORK AREA
DD1      DC  CL8'DDNAME1'           DDNAME
DSN1     DC  CL44'DATA.SET1'        DSN
ER1      DS  CL4                    RETURN CODE AREA
```

Figure 3-3. ALLOC Macro Definition - Example 1

Example 2

Operation:

Retrieve the name of the data set currently allocated to ddname DDNAME1.

Macro Definition:

```
         ALLOC FUNC=INFO,WORK=WK1,DDNAME=DD1,RETDSN=DSN1
         .................
         .................
         .................
*
WK1      DS  XL250                  REQUEST BLOCK WORK AREA
DD1      DC  CL8'DDNAME1'           DDNAME
DSN1     DC  CL44' '                RETURNED DSN
```

Figure 3-4. ALLOC Macro Definition - Example 2

CPCSNAME — Validate Name

The CPCSNAME macro is used as an inner macro to validate member/ddname names. A name is considered to be valid if:

1. It is no longer than 8 characters.
2. Its first character is alphabetic.
3. All its characters are alphanumeric.

Symbol &NAMERR, which must be defined in the outer macro, is set to 0 if the name is valid. Otherwise it is set to the number of errors detected. An MNOTE error message is generated for each error found.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede CPCSNAME.
CPCSNAME	
b	One or more blanks must follow CPCSNAME.
name	Character string.
type	Character string.

Parameters

name	Specifies the name to be validated.
type	Specifies a word to be included in the MNOTE error messages issued during the macro expansion.

Example 1

Operation:

Validate the exit routine name EXIT001.

Macro Definition:

```
         GBLA     &NAMERR              DEFINE ERROR INDICATOR SYMBOL
         GBLA     &EXNM                DEFINE EXIT NAME SYMBOL
&EXNM    SETC     'EXIT001'            INITIALIZE EXIT NAME SYMBOL
*
         CPCSNAME &EXNM,'EXITNAME'     VALID NAME ?
*
```

Figure 3-5 (Part 1 of 2). CPCSNAME Macro Definition - Example 1

```
         AIF      (&NAMERR NE 0).ERR   N. PROCESS ERROR
         AGO      .OK                  Y. CONTINUE
.ERR     DS       0H
         .................
         .................
         .................

.OK      DS       0H
         .................
         .................
         .................
```

Figure 3-5 (Part 2 of 2). CPCSNAME Macro Definition - Example 1

DKNAMODE — Test/Set Addressing Mode

The DKNAMODE macro is used to:

- Test the current address mode of a program.
- Set the address mode of a program.
- Call a program with or without address mode switch.
- Return from a program with address mode switching accomplished as required.

Functions 1 and 2 are primarily useful for RMODE=24 programs.

Functions 3 and 4 are for both RMODE=24 and RMODE=ANY programs.

CAUTION:
The contents of general registers 14 and 15 are lost (if `TYPE=CALL` **is used, the contents of registers coded in** `CALLREG=` **and** `LNKREG=` **are lost instead).**

If `NEWMODE=31` **is specified, the contents of general register 0 are lost.**

Register 1 is left intact for use as a parameter address pointer.

All addresses specified must be "clean" for the current AMODE/RMODE of the program, or unpredictable results can occur.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede DKNAMODE.
DKNAMODE	
b	One or more blanks must follow DKNAMODE.
DOC=YES DOC=NO	**Default:** MOD=NO.
TYPE=SET TYPE=TEST TYPE=CALL TYPE=RETURN	
,NEWMODE=24 ,NEWMODE=31	
,IF24=*address*	RX-type address or register (2)-(12).
,IF31=*address*	RX-type address or register (2)-(12).
,TOADDR=*address*	RX-type address or register (2)-(12).
,RC=*ret code*	Decimal number.
,CALLREG=*reg*	Decimal number (0-15). **Default:** CALLREG=15.
,LNKREG=*reg*	Decimal number (0-15). **Default:** LNKREG=14.

,RELOAD=(*reg1*,*reg2*)	Decimal number range (0-12,0-12). **Default:** RELOAD=(2,12).

Parameters

DOC=YES **DOC=NO**	If DOC=YES is specified, documentation about this macro is printed, all other parameters are ignored, and no other action is taken. If DOC=NO is specified, which is the default, the action indicated by other parameters is invoked.
TYPE=SET **TYPE=TEST** **TYPE=CALL** **TYPE=RETURN**	This is a required parameter, unless DOC=YES is specified. If TYPE=SET is specified, the AMODE indicated by the NEWMODE parameter will be in effect at the next sequential instruction. If TYPE=TEST is specified, the current addressing mode is determined, and a branch is made to the address specified by the IF24 parameter if AMODE=24 or the IF31 parameter if AMODE=31. If TYPE=CALL is specified, control is transferred to the address specified in the TOADDR= parameter. If the NEWMODE= parameter is specified, the new AMODE is set during the call (note that this could be the same as the current mode, which is not checked). The called routine is entered with register 15 (or the register specified by the CALLREG= parameter) containing the entry point, and register 14 (or the register 14 (or the register specified by the LNKREG= parameter) containing the return address. If TYPE=RETURN is specified, a return is made to the caller with mode switching as needed. Standard MVS/ESA linkages are assumed, and registers 0–12 are restored. Register 13 must be pre-loaded with the address of the previous save area. The specific registers to be restored may be indicated in the RELOAD= parameter. If the RELOAD= parameter is coded with no operands, no registers are restored.
,NEWMODE=24 **,NEWMODE=31**	Specifies the new address mode to be in effect at the next sequential instruction for TYPE=SET or at the call address for TYPE=CALL.
,IF24=*address* **,IF31=***address*	One of these parameters is required if TYPE=TEST is specified. The IF24= parameter specifies the address to branch to if AMODE=24. The IF31= parameter specifies the address to branch to if AMODE=31. Both parameters may be specified. If no parameter is specified for the current AMODE, the next sequential instruction is executed.
,TOADDR=*address*	This parameter is required if TYPE=CALL is specified. It specifies the address of the routine to be called.
,RC=*ret code*	Specifies a value to be used as a return code when the TYPE=RETURN parameter is used.
,CALLREG=*reg*	Specifies the register to hold the called address when the TYPE=CALL parameter is used. If the NEWMODE=24 parameter is specified, CALLREG= must specify an odd numbered register (the contents of register CALLREG - 1 are lost in this case).
,LNKREG=*reg*	Specifies the register to hold the return address when the TYPE=CALL parameter is specified.

,RELOAD=(*reg1,reg2)* Specifies the register(s) to be restored prior to return when the TYPE=RETURN macro is used. Standard MVS/ESA save area formats are assumed. Registers outside the range of 0–12 are considered invalid. The first register specified must be lower numbered than the second register.

Example 1

Operation:

Set addressing mode to 31.

Macro Definition:

```
         DKNAMODE   TYPE=SET,NEWMODE=31
```

Figure 3-6. DKNAMODE Macro Definition - Example 1

Example 2

Operation:

Determine addressing mode, and branch to the address contained in register 6 if AMODE=24.

Macro Definition:

```
         DKNAMODE   TYPE=TEST,IF24=(R6)
```

Figure 3-7. DKNAMODE Macro Definition - Example 2

Example 3

Operation:

Call the subroutine pointed to by register 4.

Macro Definition:

```
         DKNAMODE   TYPE=CALL,TOADDR=(4)
```

Figure 3-8. DKNAMODE Macro Definition - Example 3

DKNFILL — Fill Segment with Padding Byte

The DKNFILL macro is used to fill an area with a padding byte.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must preceed DKNFILL.
DKNFILL	
b	One or more blanks must follow DKNFILL.
start	RX-type address or register.
,*length*	RX-type address, register or constant.
,*reg1*	Even decimal number/Register equate. **Default:** R0.
,*reg2*	Even decimal number/Register equate. **Default:** R2.
,*fillbyte*	Hexadecimal number (X'xx'). **Default:** X'00'.
,REGSAVE=Y ,REGSAVE=N	**Default:** REGSAVE=Y.
,LIST=Y ,LIST=N	**Default:** LIST=Y.

Parameters

start	Specifies the address of the area to be filled with the padding byte. This is a required parameter.
length	Specifies the length of the area to be filled with the padding byte. This is a required parameter.
reg1	Specifies an even register pair to be used as work registers by the fill operation.
reg2	Specifies an even register pair to be used as work registers by the fill operation.
,*fillbyte*	Specifies the padding byte to be used in the fill operation.
,REGSAVE=Y **,REGSAVE=N**	If REGSAVE=Y is specified, the contents of the work registers are preserved.
,LIST=Y **,LIST=N**	If LIST=Y is specified, the macro expansion is included in the assembly listing.

Example 1

Operation:

Initialize the area pointed to by register 1, whose length is contained in register 3, using the register pairs R4–R5 and R2–R3 as work registers. The original work register contents are preserved. The DKNFILL macro expansion is listed.

Macro Definition:

```
        DKNFILL (R1),(R3),(R4),(R2)
```

Figure 3-9. DKNFILL Macro Definition - Example 1

Macro Expansion:

```
        PUSH   PRINT               SAVE CURRENT PRINT STATUS
        PRINT  GEN
        STM    (R4),(R4)+1,0(R1)   SAVE REGISTER OPERANDS
        STM    (R2),(R2)+1,8(R1)   ACROSS MVCL
        LR     (R4),R1             SET REG TO FILL AREA: TARGET @
        LA     (R4),16((R4))       BUMP POINTER PASSED SAVE AREA
        LR     (R4)+1,R3           TARGET LENGTH FROM REGISTER
        S      (R4)+1,=F'16'       MINUS THE SAVE AREA
        XR     (R2),(R2)           SOURCE ADDRESS
        XR     (R2)+1,(R2)+1       SOURCE LENGTH
        MVCL   (R4),(R2)           FILL SEGMENT
        LM     (R4),(R4)+1,0(R1)   RESTORE THE REGISTERS WE SAVED
        LM     (R2),(R2)+1,8(R1)
        MVI    0(R1),X'00'         INSERT THE FIRST FILL BYTE
        MVC    1(16-1,R1),0(R1)    PROPAGATE FILL BYTE
        POP    PRINT               RESTORE PREVIOUS PRINT STATUS
```

Figure 3-10. DKNFILL Macro Expansion - Example 1

DKNREGS — Generate Register Equates

The DKNREGS macro generates register equates consisting of a prefix and the register number. The prefix and registers can be specified via the macro operands.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede DKNREGS.
DKNREGS	
b	One or more blanks must follow DKNREGS.
PRE=*prefix*	Character string. **Default:** R.
,N=*reg*	Decimal number. **Default:** 15.

Parameters

PRE=*prefix* — Specifies a prefix used to build the equate register names. The register names suffix is the register number.

,N=*reg* — Specifies the highest number to be equated. Register 0 is the first register that gets equated.

Example 1

Operation:

Generate equates for registers 1–10, using the default prefix.

Macro Definition:

```
         DKNREGS N=10
```

Figure 3-11. DKNREGS Macro Definition - Example 1

Macro Expansion:

```
R0       EQU    0
R1       EQU    1
R2       EQU    2
R3       EQU    3
R4       EQU    4
R5       EQU    5
R6       EQU    6
R7       EQU    7
R8       EQU    8
R9       EQU    9
R10      EQU    10
```

Figure 3-12. DKNREGS Macro Expansion - Example 1

DKNSBT — Search SCI Sort Binary Table

The DKNSBT macro generates a routine that performs a binary search on a SCI SBTBL table, allowing assembler programs the access to this type of tables.

Input data is passed as follows:

- Register 0 points to the key entry to be searched.
- Register 1 points to the beginning of the SBTBL table.
- Register 14 has the return address.

Output data is returned as follows:

- Register 15 contains one of the following return codes:
  - **0** Direct hit (register 0 points to the matched entry)
  - **4** Within range (register 0 points to the low range)
  - **8** Outside table (register 0 has low-values)

Syntax

label	Symbol beginning in column 1.
b	One or more blanks must precede DKNSBT.
DKNSBT	
b	One or more blanks must follow DKNSBT.
WORKA=Y WORKA=N	**Default:** WORKA=N.

Parameters

WORKA=Y **WORKA=N**	If WORKA=Y is specified, the binary search routine work fields are generated.

Example 1

Operation:

Access key 03 in an SBTBL table that has 5 entries and a key of two digits.

Macro Definitions:

```
         LA    TBLEND                  COMPLETE
         STCM  R0,B'0011',TBLEND@        TABLE
         .................
         .................
         .................
*
         LA    R0,SRCHKEY              GET SEARCH KEY ADDRESS
         LA    R1,SRCHTBL              GET SBTBL TABLE ADDRESS
         L     R15,SBTRTN              GET SEARCH ROUTINE ADDRESS
         BASR  R14,R15                 CALL BINARY SEARCH ROUTINE
         LTR   R15,R15                 DIRECT HIT ?
         BZ    PROCESS                 Y. PROCESS MATCHED ENTRY
         .................
         .................
         .................
*
SBTRTN   DS    0H
         DKNSBT                        GENERATE BINARY SEARCH ROUTINE
         .................
         .................
         .................
*
         DKNSBT WORKA=Y                GENERATE SBT WORK AREAS
*
SRCHKEY  DC    X'03'                   SEARCH KEY
*
SRCHTBL  DS    0H                      SBTBL TABLE
TBLEND@  DS    AL2
         DC    H'5'
         DC    AL1(4,1)
         DC    XL4'00015700'
         DC    XL4'01015020'
         DC    XL4'03015500'
         DC    XL4'06015700'
TBLEND   DC    XL4'07015700'
```

Figure 3-13. DKNSBT Macro Definitions - Example 1

DKNXMODE — Test/Set Address Space Control Mode

The DKNXMODE macro is used to:

1. Test the current address space control (ASC) mode of a program.
2. Set the current ASC mode of a program.

CAUTION:
The contents of general registers 0, 1, 14 and 15 are lost if TYPE=TEST **is specified.**

Syntax

```
name                    Optional symbol beginning in column 1.

b                       One or more blanks must precede DKNXMODE.

DKNXMODE

b                       One or more blanks must follow DKNXMODE.

DOC=NO
DOC=YES                 Default: DOC=NO.

,TYPE=SET
,TYPE=TEST

,NEWMODE=PRIM
,NEWMODE=AR             Default: NEWMODE=PRIM.

,IFPRIM=addr            RX-type address or register.

,IFAR=addr              RX-type address or register.

,SYST=YES
,SYST=NO                Default: SYST=YES.
```

Parameters

DOC=NO
DOC=YES
If DOC=YES is specified, documentation about this macro is printed and no other action is taken.

,TYPE=SET
,TYPE=TEST
If TYPE=SET is specified, the ASC mode indicated by the NEWMODE parameter will be in effect at the next sequential instruction.

If TYPE=TEST is specified, the PSW is extracted from the current linkage stack entry, tested, and program control is routed to the address specified by either the IFAR or the IFPRIM parameter.

This is a required parameter if DOC=NO is specified.

,NEWMODE=PRIM
,NEWMODE=AR
Specifies the new ASC mode to be in effect at the next sequential instruction. It is required if TYPE=SET is specified.

,IFPRIM=*addr*
Specifies the address to receive control if the linkage stack entry PSW shows “primary mode”. Either this parameter or the IFAR parameter must be specified if TYPE=TEST is specified.

,IFAR=*addr*
Specifies the address to receive control if the linkage stack entry PSW shows “access register mode”. Either this parameter or the IFPRIM parameter must be specified if TYPE=TEST is specified.

,SYST=YES **,SYST=NO**	If SYST=YES is specified, the SYSTSTATE macro is issued for ASC mode changes. This is an optional parameter when TYPE=SET is specified.

Example 1

Operation:

Set the ASC mode to access register, issuing the SYSSTATE macro.

Macro Definition:

```
         DKNXMODE  TYPE=SET,NEWMODE=AR,SYST=YES
```

Figure 3-14. DKNXMODE Macro Definition - Example 1

Example 2

Operation:

Test the ASC mode, and branch to label PRASCMDE if in prime space.

Macro Definition:

```
         DKNXMODE  TYPE=TEST,IFPRIM=PRASCMDE
         .................
         .................
         .................
*
PRASCMDE DS  0H
         .................
         .................
         .................
```

Figure 3-15. DKNXMODE Macro Definition - Example 2

FFLDL — Generate Field Length Equates/Constants

The FFLDL macro is used to generate equated labels and half word constants equal to the length of the compressed codeline record fields.

Syntax

```
name                    Optional symbol beginning in column 1.

b                       One or more blanks must precede FFLDL.

FFLDL

b                       One or more blanks must follow FFLDL.

GENEQU=YES
GENEQU=NO               Default: GENEQU=NO.

,GENPRE=YES
,GENPRE=NO              Default: GENPRE=NO.

,PREF=prefix            3-character string.
                        Default: FMD.
```

Parameters

GENEQU=YES
GENEQU=NO
If GENEQU=YES is specified, equates for the lengths of the codeline record fields are generated. The equated labels consist of a prefix (specified by the PREF parameter) and a field mnemonic.

,GENPRE=YES
,GENPRE=NO
If GENPRE=YES is specified, half word constants equal to the lengths of the codeline record fields are generated. The constant labels consist of a prefix (specified by the PREF parameter) and a field (specified by the PREF parameter) and a field mnemonic.

,PREF=*prefix*
Specifies a 3-character prefix to be included in the equate and constant labels generated by the FFLDL macro.

Example 1

Operation:

Generate field length equates for codeline records defined as:

```
L'DIAMT  = 5
L'DIPCTL = 3
L'DIONUS = 7
L'DIOPT1 = 0
L'DIABA  = 4
L'DIRET  = 1
L'DIAUX  = 5
```

FMD is used as the default label prefix.

Macro Definition:

```
FFLDL   GENEQU=YES,PREF=WRK
```

Figure 3-16. FFLDL Macro Definition - Example 1

Macro Expansion:

```
FMDAMTL  EQU   5                      LENGTH OF AMOUNT
FMDPCTLL EQU   3                               PROCESS CONTROL
FMDONUSL EQU   7                               ONUS
FMDOPT1L EQU   0                               OPTIONAL FIELD 1
FMDABAL  EQU   4                               ABA
FMDRETL  EQU   1                               RETURN ITEM
FMDAUXL  EQU   5                               AUXILIARY
FMDSEQL  EQU   6                               SEQUENCE NUMBER
FMDF14L  EQU   0                               FIELD 14
FMDF15L  EQU   0                               FIELD 15
```

Figure 3-17. FFLDL Macro Expansion - Example 1

Example 2

Operation:

Generate field length equates and constants for codeline records defined as:

```
L'DIAMT  = 5
L'DIPCTL = 2
L'DIONUS = 7
L'DIOPT1 = 0
L'DIABA  = 4
L'DIRET  = 3
L'DIAUX  = 5
```

WRK is used as label prefix.

Macro Definition:

```
         FFLDL   GENEQU=YES
```

Figure 3-18. FFLDL Macro Definition - Example 2

Macro Expansion:

```
WRKAMTL  EQU   5                      LENGTH OF AMOUNT
WRKPCTLL EQU   2                               PROCESS CONTROL
WRKONUSL EQU   7                               ONUS
WRKOPT1L EQU   0                               OPTIONAL FIELD 1
WRKABAL  EQU   4                               ABA
WRKRETL  EQU   3                               RETURN ITEM
WRKAUXL  EQU   5                               AUXILIARY
WRKSEQL  EQU   6                               SEQUENCE NUMBER
WRKF14L  EQU   0                               FIELD 14
WRKF15L  EQU   0                               FIELD 15
```

Figure 3-19 (Part 1 of 2). FFLDL Macro Expansion - Example 2

```
WRKAMTP  DC    AL2(5)                    AMOUNT LENGTH
WRKPCTLP DC    AL2(2)                    PROCESS CONTROL LENGTH
WRKONUSP DC    AL2(7)                    ONUS LENGTH
WRKOPT1P DC    AL2(0)                    OPT FIELD 1 NOT DEFINED
WRKABAP  DC    AL2(4)                    ABA LENGTH
WRKRETP  DC    AL2(3)                    FLD 6 LENGTH
WRKAUXP  DC    AL2(5)                    AUXILIARY LENGTH
WRKSEQP  DC    AL2(6)                    SEQ NUMBER LENGTH
WRKF14P  DC    AL2(0)                    FIELD 14 NOT DEFINED
WRKF15P  DC    AL2(0)                    FIELD 15 NOT DEFINED
```

Figure 3-19 (Part 2 of 2). FFLDL Macro Expansion - Example 2

FPACK — Pack Unsigned a Zoned Numeric Field

The FPACK macro is used to pack zoned numeric fields into unsigned packed fields.

Any length field may be packed. The FPACK macro generates the necessary number of PACK instructions to properly pack the number.

The FFLDL macro is used as an inner macro to determine codeline record field lengths as specified by the MDX macro.

CAUTION:
When packing into a record where the fields are contiguous, perform all operations from left to right within the record to avoid overlaying good data with the pad character.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede FPACK.
FPACK	
b	One or more blanks must follow FPACK.
target	RX-type address.
,*source*	RX-type address.
,FLD=*number* ,FLD=AMT ,FLD=PCTL ,FLD=ONUS ,FLD=OPT1 ,FLD=ABA ,FLD=RET ,FLD=AUX ,FLD=SEQ	Decimal number.
,PAD=*number*	*Hexadecimal number.* **Default:** *00, if FLD = PCTL.* *AA, if FLD not = PCTL.*

Parameters

target	Specifies the target field for the pack operation. The length of the field must not be specified.
source	Specifies the source field for the pack operation. The length of the field must not be specified.
,FLD=*number* **,FLD=AMT** **,FLD=PCTL** **,FLD=ONUS** **,FLD=OPT1** **,FLD=ABA** **,FLD=RET** **,FLD=AUX** **,FLD=SEQ**	Specifies an codeline record field whose packed length is to be used as the length of the pack operation. Valid numeric operands are 1–8. Valid character mnemonics are AMT, PCTL, ONUS, OPT1, ABA, RET, AUX or SEQ. If this parameter is not specified, the length of the pack operation is determined from the target operand. This parameter is required if the target operand address is specified as OFFSET(REGISTER).

,PAD=*number* Specifies two hex characters that overlay the extra byte generated by the pack operation.

Example 1

Operation:

Convert a 30-byte zone numeric string to an unsigned packed 16-byte number, using FA for the extra byte pad, and a pack length equal to the MDX defined compressed on-us length (15).

Macro Definition:

```
         FPACK   TARGET1,SOURCE1,FLD=ONUS,PAD=FA
         ..............
         ..............
         ..............
*
TARGET1  DC  XL16'00000000000000000000000000000000'
SOURCE1  DC  CL41'12345678901234567890123456789O'
```

Figure 3-20. FPACK Macro Definition - Example 1

Macro Expansion:

```
         PACK  TARGET(8),SOURCE(15)
         PACK  TARGET+7(8),SOURCE+14(15)
         PACK  TARGET+14(2),SOURCE+28(3)
         MVI   TARGET+15,X'FA'
```

Figure 3-21. FPACK Macro Expansion - Example 1

Result:

```
TARGET1  DC  XL16'12345678901234567890123456789OFA'
```

Figure 3-22. FPACK Macro Result - Example 1

Example 2

Operation:

Convert a 20-byte zone numeric string to a packed unsigned number, using AA for the extra byte pad.

Macro Definition:

```
         FPACK   TARGET1,SOURCE1
         ................
         ................
         ................
*
TARGET1  DC  XL18'000000000000000000000000000000000000'
         DC  XL1'00'
SOURCE1  DC  CL20'12345678901234567890'
```

Figure 3-23. FPACK Macro Definition - Example 2

Macro Expansion:

```
         PACK  TARGET1(8),SOURCE1(15)
         PACK  TARGET1+7(8),SOURCE1+14(15)
         PACK  TARGET1+14(5),SOURCE1+28(9)
         MVI   TARGET1+18,X'AA'
```

Figure 3-24. FPACK Macro Expansion - Example 2

Result:

```
TARGET1  DC  XL18'12345678901234567890000000000000000000'
         DC  XL1'AA'
```

Figure 3-25. FPACK Macro Result - Example 2

FUNPK — Unpack a Packed Unsigned Field

The FUNPK macro is used to unpack packed unsigned fields into zoned numeric fields.

Any length field may be unpacked. The FUNPK macro generates the necessary number of UNPK instructions to properly unpack the number.

The FFLDL macro is used as an inner macro to determine codeline record field lengths as specified by the MDX macro.

Syntax

name	Optional symbol beginning in column 1.
b	One or more blanks must precede FUNPK.
FUNPK	
b	One or more blanks must follow FUNPK.
target	RX-type address.
,*source*	RX-type address.
,FLD=*number* ,FLD=AMT ,FLD=PCTL ,FLD=ONUS ,FLD=OPT1 ,FLD=ABA ,FLD=RET ,FLD=AUX ,FLD=SEQ	Decimal number.
,PAD=*number*	*Hexadecimal number.* **Default:** *40.*

Parameters

target	Specifies the target field for the unpack operation. The length of the field must not be specified.
source	Specifies the source field for the unpack operation. The length of the field must not be specified.
,FLD=*number* **,FLD=AMT** **,FLD=PCTL** **,FLD=ONUS** **,FLD=OPT1** **,FLD=ABA** **,FLD=RET** **,FLD=AUX** **,FLD=SEQ**	Specifies a codeline record field whose packed length is to be used as the length of the unpack operation. Valid numeric operands are 1–8. Valid character mnemonics are AMT, PCTL, ONUS, OPT1, ABA, RET, AUX or SEQ. If this parameter is not specified, the length of the unpack operation is determined from the source operand. This parameter is required if the source operand address is specified as OFFSET(REGISTER).
PAD=*number*	Specifies two hex characters that overlay the extra byte generated by the unpack operation.

Example 1

Operation:

Convert a 15-byte packed unsigned field to a 30-byte zone numeric, using C1 for the extra byte pad, and an unpack length equal to the MDX defined compressed on-us length (15).

Macro Definition:

```
         FUNPK   TARGET1,SOURCE1,FLD=ONUS,PAD=C1
         ................
         ................
         ................
*
TARGET1  DC  CL31'                                '
SOURCE1  DC  XL15'123456789012345678901234567890'
```

Figure 3-26. FUNPK Macro Definition - Example 1

Macro Expansion:

```
         UNPK  TARGET1(15),SOURCE1(8)
         UNPK  TARGET1+14(15),SOURCE1+7(8)
         UNPK  TARGET1+28(3),SOURCE1+14(2)
         MVI   TARGET1+30,X'C1'
```

Figure 3-27. FUNPK Macro Expansion - Example 1

Result:

```
TARGET1  DC  CL31'12345678901234567890123456789OA'
```

Figure 3-28. FUNPK Macro Result - Example 1

Example 2

Operation:

Convert a 5-byte packed unsigned number to zoned numeric, using 40 for the extra byte pad.

Macro Definition:

```
         FUNPK   TARGET1,SOURCE1
         ................
         ................
         ................
*
TARGET1  DC  CL15'000000000000000'
SOURCE1  DC  XL5'1234567890'
```

Figure 3-29. FUNPK Macro Definition - Example 2

Macro Expansion:

```
         UNPK  TARGET1(11),SOURCE1(6)
         MVI   TARGET1+10,X'40'
```

Figure 3-30. FUNPK Macro Expansion - Example 2

Result:

```
TARGET1  DC  CL15'1234567890 0000'
```

Figure 3-31. FUNPK Macro Result - Example 2

XRETURN — Perform Program Exit Linkage

This macro is used to return control to the calling program and signal normal termination of the called program.

This macro is almost identical to the RETURN MVS/ESA macro, except for the last instruction of its expansion, which is a BSM 0,14 instead of a BR 14. XRETURN differs from RETURN in that it re-establishes the calling program's AMODE.

Syntax

```
name                    Optional symbol beginning in column 1.

b                       One or more blanks must precede XRETURN.

XRETURN

b                       One or more blanks must follow XRETURN.

(reg)                   Decimal number.
(reg1,reg2)             Decimal number range (14-12,14-12).

,T

,RC=ret code            Decimal number.
,RC=(15)
```

Parameters

(*reg***)**
(*reg1,reg2***)**
Specifies the register or range of registers to be restored from the save area pointed to by the address in register 13.

,T
This parameter causes the flagging of the save area used by the called program. The low-order bit of word 4 of the save area is set to 1 after the registers have been loaded; this indicates that a called program has executed a return to its caller.

,RC=*ret code*
,RC=(15)
Specifies the return code to be passed to the calling program. If a decimal number is coded, the return code is placed right-adjusted in register 15 before return is made; if register 15 is coded, the return code has been previously loaded into register 15 and the contents of register 15 are not altered or restored from the save area.

Example 1

Operation:

Restore registers 14, 15 and 0–12, flag the save area, and return with a code of 0.

Macro Definition:

```
         XRETURN  (14,12),T,RC=0
```

Figure 3-32. XRETURN Macro Definition - Example 1

Example 2

Operation:

Restore registers 2–12, and return with a code already present in register 15.

Macro Definition:

```
         XRETURN  (2,12),RC=(15)
```

Figure 3-33. XRETURN Macro Definition - Example 2

Example 3

Operation:

Restore registers 14, 15 and 0–12, and return.

Macro Definition:

```
         XRETURN  (14,12)
```

Figure 3-34. XRETURN Macro Definition - Example 3

Chapter 4. Subroutines Provided by CPCS-I

Overview 4-3
DKNDATE — Perform Date/Time Functions 4-4
DKNDEQ — Dequeue on Resource Name 4-9
DKNDYNA — Dynamically Allocate/Unallocate Permanent Data Sets 4-11
DKNENQ — Enqueue on Resource Name 4-15
DKNPDSIO — Perform PDS I/O Processing 4-17
DKNQGET — Perform QSAM Input Processing 4-23
DKNQPUT — Perform QSAM Output Processing 4-28
DKNTDYNA — Dynamically Allocate/Unallocate Temporary Data Sets 4-33
DKNVSMIO — Perform VSAM I/O Processing 4-37

Overview

This chapter documents general-use CPCS-I subroutines that may be called by CPCS-I application programs and exit routines. Most of these subroutines may also be invoked in a batch environment.

See the copybooks for the control blocks used to interface with the CPCS-I subroutines documented in this chapter.

DKNDATE — Perform Date/Time Functions

DKNDATE may be called to:

- Acquire the current date in one of the valid CPCS-I date formats (see the MDEF DATE parameter in the *CPCS-I Customization Guide*).
- Acquire the current time.
- Convert a date to a different valid CPCS-I format.
- Validate a date.

Linkage

The standard MVS/ESA linkage conventions are followed.

DKNDATE is a CPCS-I resident routine. Field ADKNDATE, in the CPCS-I parameter list, contains the module's entry point address. Information is passed in the following control block:

Table 4-1. DKNDATE Communication Control BLock

Control Block	COBOL Copybook	Assembler Copybook
DKNDATE Call Parameters	DKNCRDA	DKNDATEP

Example 1

Operation:

Obtain the current date in the CPCS-I default format, along with the current time in the hh.mm.ss format. Verify field SP-DATE contains a valid packed date with a century indicator.

COBOL Code:

```
  ID DIVISION.
  PROGRAM-ID. TSDATE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
 /
  WORKING-STORAGE SECTION.

  01  WS-CPCS-DATE               PIC  X(10).
  01  WS-CPCS-TIME               PIC  X(10).
  01  WS-DATE-RC-9               PIC  9(01).
  01  WS-DATE-MSG                PIC  X(30).

  COPY DKNCRDA.
 /
  LINKAGE SECTION.
```

Figure 4-1 (Part 1 of 3). DKNDATE Subroutine Example 1

```
 COPY DKNCATCB.

 01  START-PARMS.
     05                          PIC  X(10).
     05  SP-DATE                 PIC S9(07)  COMP-3.
     05                          PIC  X(54).
/
 PROCEDURE DIVISION              USING APTCB
                                       START-PARMS.

     SET  CPU-TO-DEFAULT         TO TRUE.
     MOVE APTCB-PARM-X           TO DATEPRML.

     CALL 'DKNDATE'              USING DATEPARM.

     IF  NOT DA-GOOD-RETURN
         MOVE DATE-MSG           TO WS-DATE-MSG
         MOVE DATE-RC            TO WS-DATE-RC-9
         MOVE WS-DATE-RC-9       TO RETURN-CODE

         ..................
         ..................
         ..................

     ELSE
         MOVE DATEDISP               TO WS-CPCS-DATE
         MOVE TIMEDISP               TO WS-CPCS-TIME

         ..................
         ..................
         ..................

         SET  PACKED-TO-DEFAULT      TO TRUE
         MOVE APTCB-PARM-X           TO DATEPRML

         CALL 'DKNDATE'              USING DATEPARM

         IF  NOT DA-GOOD-RETURN
             MOVE DATE-MSG           TO WS-DATE-MSG
             MOVE DATE-RC            TO WS-DATE-RC-9
             MOVE WS-DATE-RC-9       TO RETURN-CODE

             ..................
             ..................
             ..................
```

Figure 4-1 (Part 2 of 3). DKNDATE Subroutine Example 1

```
         ELSE

             ...................
             ...................
             ...................

     GOBACK.
```

Figure 4-1 (Part 3 of 3). DKNDATE Subroutine Example 1

Example 2

Operation:

Obtain the current date in the YYYY.DDD format. Convert the date in field SPDATE from the MM/DD/YYYY format to the DD/MM/YYYY format.

Assembler Code:

```
TSDATE    CSECT
TSDATE    AMODE 31
TSDATE    RMODE ANY
*
          SAVE  (14,12)            SAVE REGISTERS
          BASR  R12,0              GET ROUTINE @
          USING *,R12              BASE SUBROUTINE
*
          LA    R14,SAVEAREA       GET OUR SAVE AREA @
          ST    R14,8(R13)         STORE FORWARD SAVE AREA @
          ST    R13,4(R14)         STORE BACKWARD SAVE AREA @
          LR    R13,R14            GET NEW SAVE AREA @
*
          L     R6,0(R1)           GET APTCB @
          USING APTCB,R6           MAP APTCB
          L     R7,4(R1)           GET START PARMS @
          USING STPARMS,R7         MAP START PARMS
```

Figure 4-2 (Part 1 of 3). DKNDATE Subroutine Example 2

```
*
         MVI   DATEFC,CPU_TO_YYYYDDD  SET REQUEST CODE
         XC    DATESTOR,DATESTOR  CLEAR WORK STORAGE @
         L     R5,TCBPARM         GET CPCS-I PARM LIST @
         ST    R5,DATEPRML        PARM LIST @ TO DATE PARMS
         USING PARMLST,R5         MAP CPCS-I PARM LIST
         L     R15,ADKNDATE       GET DATE @
         LA    R1,DATEPARM        GET DATE PARMS @
         ST    R1,DATEPRM         DATE PARMS @ TO DATE PARMS
         LA    R1,DATEPRM         GET DATE PARMS @
         BASSM R14,R15            CALL DATE
         CLI   DATERC,DATERCG     SUCCESSFUL COMPLETION ?
         BE    CVTDATE            Y. CONTINUE
         MVC   WKMSG,DATEEMSG     GET DATE ERROR MESSAGE
         MVC   WKRCD,DATERCD      GET DATE RETURN CODE
         ..............
         ..............
         ..............
         B     RETURN             EXIT PROGRAM
*
CVTDATE  MVC   WKDATE,DATEDISP    SAVE RETURNED DATE
         ..............
         ..............
         ..............
*
         MVI   DATEFC,MMDDYYYY_TO_DDMMYYYY  SET REQUEST CODE
         MVC   DATEIN,SPDATE      SET OLD FORMAT DATE
         XC    DATESTOR,DATESTOR  CLEAR WORK STORAGE @
         ST    R5,DATEPRML        PARM LIST @ TO DATE PARMS
         L     R15,ADKNDATE       GET DATE @
         LA    R1,DATEPARM        GET DATE PARMS @
         ST    R1,DATEPRM         DATE PARMS @ TO DATE PARMS
         LA    R1,DATEPRM         GET DATE PARMS @
         BASSM R14,R15            CALL DATE
         CLI   DATERC,DATERCG     SUCCESSFUL COMPLETION ?
         BE    USEDATE            Y. CONTINUE
         MVC   WKMSG,DATEEMSG     GET DATE ERROR MESSAGE
         MVC   WKRCD,DATERCD      GET DATE RETURN CODE
         ..............
         ..............
         ..............
         B     RETURN             EXIT PROGRAM
*
USEDATE  ..............
         ..............
         ..............
*
RETURN   L R13,SAVEAREA+4         RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15) RETURN
*
         DKNREGS
*
WKMSG    DS    CL30               DATE MESSAGE SAVE AREA
WKRCD    DS    C                  DATE RETURN CODE SAVE AREA
*
SAVEAREA DS    18F                OUR SAVE AREA
```

Figure 4-2 (Part 2 of 3). DKNDATE Subroutine Example 2

```
*
         COPY DKNAPTCB            APTCB
*
STPARMS  DSECT                    PASSED PARMS
         DS    CL20
SPDATE   DS    CL10               DATE TO BE CONVERTED
         DS    CL58
*
         END
```

Figure 4-2 (Part 3 of 3). DKNDATE Subroutine Example 2

DKNDEQ — Dequeue on Resource Name

DKNENQ performs resource name dequeuing, within the CPCS-I job address space, for COBOL programs.

Note: See the MVS/ESA library for information on the DEQ macro and its return codes.

Linkage

Information is passed in the following interface control blocks:

Table 4-2. DKNDEQ Communication Control Blocks

Control Block	COBOL Copybook
QNAME	
RNAME • The first byte contains the binary length (1–255) of the QNAME specified in the following bytes.	
Work Area • This work area, used by DKNDEQ, must have a size of 25 bytes.	

DKNDEQ returns the DEQ macro completion code in the RETURN-CODE special register.

Restrictions

The RET=NONE DEQ request type is not supported.

Example 1

Operation:

Call DKNDEQ to release control of a resource, dequeuing on QNAME QNAME1 and RNAME RNAME1.

COBOL Code:

```
      ...................
      ...................
      ...................
/
 WORKING-STORAGE SECTION.
```

Figure 4-3 (Part 1 of 2). DKNDEQ Subroutine Example 1

```
     ...................
     ...................
     ...................

 01  DEQ-QNAME                        PIC X(008) VALUE 'QNAME1'.

 01  DEQ-RNAME.
     05  ENQ-RN-LENGTH                PIC X(001) VALUE X'06'.
     05  ENQ-RN-RNAME                 PIC X(006) VALUE 'RNAME1'.

 01  ENQ-WORK-AREA                    PIC X(025).

 01  REQUEST-COMPLETED                PIC 9(002) VALUE 0.

     ...................
     ...................
     ...................
/
 PROCEDURE DIVISION.

     ...................
     ...................
     ...................

     CALL 'DKNDEQ'                     USING DEQ-QNAME
                                             DEQ-RNAME
                                             DEQ-WORK-AREA.

     IF RETURN-CODE                    NOT = REQUEST-COMPLETED
         PERFORM 1000-PROCESS-RETURN-CODE

     ...................
     ...................
     ...................

 1000-PROCESS-RETURN-CODE.

     ...................
     ...................
     ...................
```

Figure 4-3 (Part 2 of 2). DKNDEQ Subroutine Example 1

DKNDYNA — Dynamically Allocate/Unallocate Permanent Data Sets

DKNDYNA may be called to dynamically allocate and unallocate permanent disk and tape data sets.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNDYNA:

Table 4-3. DKNDYNA Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Request Code • This is a three character field. Valid codes are: **ALC** Allocate data set. **UNA** Unallocate data set.		
ddname • This is an eight character field that contains the ddname associated to the data set in both the application program and the DKNDSAT table.		
Application Task Control Block	DKNCATCB	DKNAPTCB
Parameters End Indicator • This is a one byte field that always contains the value X'FF'.		

DKNDYNA may return the following completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs):

Table 4-4. DKNDYNA Return Codes

Code (hex)	Description
+00	Successful Completion.
+01	Successful completion for an absolute GDG allocation request.
+04	Allocation failed with a return code of 4 and a reason code of X'0484' (request cancelled by operator). Unallocation failed.
+08	Allocation failed with a return code of 4 and a reason code other than X'0484'.

DKNDYNA abends with a user code of 3 if the dynamic allocation function returns a completion code other than 0 or 4.

A supervisor message, containing the return and reason codes, is issued by DKNDYNA whenever a non-zero completion code is returned by the MVS/ESA dynamic allocation function. See the MVS/ESA library for a description of the dynamic allocation function return and reason codes.

Restrictions

To be eligible to use this feature, data sets must be defined in the DKNDSAT table. See “Dynamically Allocating Data Sets” in the *CPCS-I Customization Guide* for information on how to create DKNDSAT table entries.

Example 1

Operation:

Allocate, process and unallocate the cataloged data set CPCS.DSNAME1. It is assumed that the DKNDSAT table contains an entry with the parameters DYNDEV=DISK, DYNDDN=DDNAME1, DYNDSN=CPCS.DDNAME1, DYNSTAT=SHR, DYNNORM=KEEP, DYNCOND=KEEP and DYNUNIT=XXXX, where XXXX is a valid installation disk unit number.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSDYNA.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  DYNA-PARMS.
     05  DP-REQUEST-CODE        PIC X(03).
         88  DP-ALLOCATE                  VALUE 'ALC'.
         88  DP-UNALLOCATE                VALUE 'UNA'.
     05  DP-DDNAME              PIC X(08)  VALUE 'DDNAME1'.
     05  DP-ENDPARM             PIC X(01)  VALUE X'FF'.

 01  SUCCESSFUL-OPERATION       PIC 9(01)  VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC X(68).
/
 PROCEDURE DIVISION             USING APTCB
                                      START-PARMS.

     PERFORM 1000-ALLOCATE-DATA-SET.

     IF  RETURN-CODE            = SUCCESSFUL-OPERATION
         PERFORM 2000-PROCESS-DATA-SET
         PERFORM 3000-UNALLOCATE-DATA-SET.

     GOBACK.

 1000-ALLOCATE-DATA-SET.

     SET DP-ALLOCATE            TO TRUE.
```

Figure 4-4 (Part 1 of 2). DKNDYNA Example 1

```
     CALL 'DKNDYNA'            USING DP-REQUEST-CODE
                                     DP-DDNAME
                                     APTCB
                                     DP-ENDPARM.

 2000-PROCESS-DATA-SET.

     ...................
     ...................
     ...................

 3000-UNALLOCATE-DATA-SET.

     SET DP-UNALLOCATE         TO TRUE.

     CALL 'DKNDYNA'            USING DP-REQUEST-CODE
                                     DP-DDNAME
                                     APTCB
                                     DP-ENDPARM.
```

Figure 4-4 (Part 2 of 2). DKNDYNA Example 1

Example 2

Operation:

Allocate, create and unallocate a new generation of the GDG data set CPCS.DSNAME1. It is assumed that the DKNDSAT table contains an entry with the parameters DYNDEV=DISK, DYNDDN=DDNAME1, DYNDSN=CPCS.DDNAME1(+1), DYNDCB=DSNAME1.MODEL, DYNSTAT=NEW, DYNNORM=CATLG, DYNCOND=KEEP, DYNSTYP=TTT, DYNPSPA=P, DYNSSPA=S and DYNUNIT=XXXX, where DSNAME1.MODEL is a DSCB model, TTT is the allocation space units, P is the primary space, S is the secondary space and XXXX is a valid installation disk unit number.

Assembler Code:

```
TSDYNA   CSECT
TSDYNA   AMODE 31
TSDYNA   RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
```

Figure 4-5 (Part 1 of 2). DKNDYNA Example 2

```
         MVC   APTCB@,0(R1)          GET APTCB @
         MVC   DPREQCDE,ALLO         SET 'ALLOCATE' REQUEST CODE
         LA    R1,PARMS              GET DKNDYNA PARM LIST @
         L     R15,DYNA@             GET DKNDYNA @
         BASSM R14,R15               CALL DKNDYNA TO ALLOCATE
         CH    R15,RETCODE1          SUCCESSFUL OPERATION ?
         BNE   DYNAERR               N. PROCESS DKNDYNA ERROR
*
         LA    R15,0                 RESET RETURN CODE
         .................           DATA SET'S OPEN, WRITES & CLOSE
         .................
         .................
*
         MVC   DPREQCDE,UNALLO       SET 'UNALLOCATE' REQUEST CODE
         LA    R1,PARMS              GET DKNDYNA PARM LIST @
         L     R15,DYNA@             GET DKNDYNA @
         BASSM R14,R15               CALL DKNDYNA TO ALLOCATE
         LTR   R15,R15               SUCCESSFUL OPERATION ?
         BZ    RETURN                Y. EXIT PROGRAM
*
DYNAERR  DS   0H                     PROCESS DKNDYNA ERROR
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
DYNA@    DC    V(DKNDYNA)            DKNDYNA @
*
RETCODE1 DC    H'1'                  SUCCESSFUL GDG ALLOCATION
*
PARMS    DS   0F                     DKNDYNA PARM LIST
         DC    A(DPREQCDE)           . REQUEST CODE FIELD @
         DC    A(DDNAME1)            . DDNAME @
APTCB@   DS    A                     . APTCB @
         DC    A(ENDPARMS)           . END OF PARMS INDICATOR
*
DPREQCDE DS    CL3                   REQUEST CODE
ALLO     DC    C'ALC'                . ALLOCATE DATA SET
UNALLO   DC    C'UNA'                . UNALLOCATE DATA SET
*
DDNAME1  DC    CL8'DDNAME1'          DATA SET ASSOCIATED DDNAME
*
ENDPARMS DC    X'FF'                 END OF PARMS INDICATOR
*
SAVEAREA DS  18F                     OUR SAVE AREA
*
         END
```

Figure 4-5 (Part 2 of 2). DKNDYNA Example 2

DKNENQ — Enqueue on Resource Name

DKNENQ performs exclusive resource name enqueuing, within the CPCS-I job address space, for COBOL programs.

Note: See the MVS/ESA library for information on the ENQ macro.

Linkage

Information is passed in the following interface control blocks:

Table 4-5. DKNENQ Communication Control Blocks

Control Block	COBOL Copybook
QNAME	
RNAME • The first byte contains the binary length (1–255) of the QNAME specified in the following bytes.	
ENQ Request Type	DKNENQC
Work Area • This work area, used by DKNENQ, must have a size of 25 bytes.	

DKNENQ returns the ENQ macro completion code in the RETURN-CODE special register. See the MVS/ESA library for information on the ENQ macro return codes.

Restrictions

The RET=CHNG ENQ request type is not supported.

Example 1

Operation:

Call DKNENQ to request unconditional control of a resource, enqueuing on QNAME QNAME1 and RNAME RNAME1.

COBOL Code:

```
      ...................
      ...................
      ...................
 /
  WORKING-STORAGE SECTION.
```

Figure 4-6 (Part 1 of 2). DKNENQ Subroutine Example 1

```
     ...................
     ...................
     ...................

 01  ENQ-QNAME                        PIC X(008) VALUE 'QNAME1'.

 01  ENQ-RNAME.
     05  ENQ-RN-LENGTH                PIC X(001) VALUE X'06'.
     05  ENQ-RN-RNAME                 PIC X(006) VALUE 'RNAME1'.

 01  ENQ-WORK-AREA                    PIC X(025).

 COPY DKNENQC.

 01  REQUEST-COMPLETED                PIC 9(002) VALUE 0.

     ...................
     ...................
     ...................
/
 PROCEDURE DIVISION.

     ...................
     ...................
     ...................

     SET ENQ-RET-NONE                  TO TRUE.

     CALL 'DKNENQ'                     USING ENQ-QNAME
                                             ENQ-RNAME
                                             ENQ-REQUEST-TYPE
                                             ENQ-WORK-AREA.

     IF RETURN-CODE                    NOT = REQUEST-COMPLETED
         PERFORM 1000-PROCESS-RETURN-CODE

     ...................
     ...................
     ...................

 1000-PROCESS-RETURN-CODE.

     ...................
     ...................
     ...................
```

Figure 4-6 (Part 2 of 2). DKNENQ Subroutine Example 1

DKNPDSIO — Perform PDS I/O Processing

DKNPDSIO may be called to:

- Read existing PDS members.
- Create new PDS members.

An “open” call must be issued before making any read/write calls or start processing a new member.

A single record is processed on each read/write call.

A “close” call must be issued when all records have been processed, except when in input mode, if the end-of-file condition has been reached.

The PDS does not need to be defined (no assembler DCBs or COBOL FDs are required) in the programs using the DKNPDSIO services.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 4-6. DKNPDSIO Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNPDSIO Call Parameters • The last two fields must contain binary zeroes before the first call is issued.	DKNPDSC	DKNPDSA
Record I/O Area		

DKNPDSIO may return the following completion codes in the call parameters:

Table 4-7. DKNPDSIO Return Codes

Code (dec)	Description
+00	Request completed.
+04	Member not found.
+08	I/O error.
+12	Open/close error.
+16	End-of-file detected.
+20	PDS not opened for the requested member.
+24	PDS format error.
+28	Invalid request code.
+36	Directory update error.
+40	Member already exists on an output “open” request.
+44	Member in use/ddname not allocated.

Restrictions

From one call parameter area, a calling program can have only one PDS open at one time and only one member from any one PDS.

The referenced ddname must be allocated to the CPCS-I job.

Example 1

Operation:

Read members MEMBER1 and MEMBER2 from the PDS allocated to ddname DDA, copying records having a “T” in the first byte, into a new member MEMBER3 on the PDS allocated to ddname DDB.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSPDSIO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  DDNAME-A                      PIC X(08) VALUE 'DDA'.
 01  DDNAME-B                      PIC X(08) VALUE 'DDB'.

 01  MEMBER-1                      PIC X(08) VALUE 'MEMBER1'.
 01  MEMBER-2                      PIC X(08) VALUE 'MEMBER2'.
 01  MEMBER-3                      PIC X(08) VALUE 'MEMBER3'.

 01  I-O-AREA.
     05  IO-RECORD-CODE            PIC X(01).
         88  IO-TRANSFER                    VALUE 'T'.
     05  FILLER                    PIC X(79).

 COPY DKNPDSC                       REPLACING ==:TAG:==
                                    BY        ==DDA==.

 COPY DKNPDSC                       REPLACING ==:TAG:==
                                    BY        ==DDB==.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  TCBSPARM                      PIC X(68).
/
 PROCEDURE DIVISION                 USING APTCB
                                          TCBSPARM.

     MOVE MEMBER-1                  TO PDS-DDA-MEMBER.
     PERFORM 1000-OPEN-DDA.
```

Figure 4-7 (Part 1 of 3). DKNPDSIO Subroutine Example 1

```
    IF  PDS-DDA-GOOD
        PERFORM 2000-OPEN-DDB

        IF PDS-DDB-GOOD
            PERFORM 3000-PROCESS-MEMBER
                UNTIL NOT PDS-DDA-GOOD
                  OR  NOT PDS-DDB-GOOD

            IF  PDS-DDA-EOF
                SET PDS-DDA-GOOD   TO TRUE

                IF  PDS-DDB-GOOD
                    MOVE MEMBER-2  TO PDS-DDA-MEMBER
                    PERFORM 1000-OPEN-DDA

                    IF  PDS-DDA-GOOD
                        PERFORM 3000-PROCESS-MEMBER
                            UNTIL NOT PDS-DDA-GOOD
                            OR  NOT PDS-DDB-GOOD

                            IF  PDS-DDA-EOF
                                SET PDS-DDA-GOOD   TO TRUE

                                IF  PDS-DDB-GOOD
                                    PERFORM 6000-CLOSE-DDB.

    IF  NOT PDS-DDA-GOOD
     OR NOT PDS-DDB-GOOD
        PERFORM 7000-PROCESS-PDSIO-ERROR.

    GOBACK.

1000-OPEN-DDA.

    MOVE DDNAME-A                 TO PDS-DDA-DDNAME.
    SET  PDS-DDA-OPEN-IN          TO TRUE.

    CALL 'DKNPDSIO'               USING PDS-DDA-PARM-LIST
                                        I-O-AREA.

2000-OPEN-DDB.

    MOVE DDNAME-B                 TO PDS-DDB-DDNAME.
    MOVE MEMBER-3                 TO PDS-DDB-MEMBER.
    SET  PDS-DDB-OPEN-OUT         TO TRUE.

    CALL 'DKNPDSIO'               USING PDS-DDB-PARM-LIST
                                        I-O-AREA.

3000-PROCESS-MEMBER.

    PERFORM 4000-READ-DDA.
```

Figure 4-7 (Part 2 of 3). DKNPDSIO Subroutine Example 1

```
    IF  PDS-DDA-GOOD
        IF  IO-TRANSFER
            PERFORM 5000-WRITE-DDB.

4000-READ-DDA.

    SET PDS-DDA-READ              TO TRUE.

    CALL 'DKNPDSIO'               USING PDS-DDA-PARM-LIST
                                        I-O-AREA.

5000-WRITE-DDB.

    SET PDS-DDB-WRITE             TO TRUE.

    CALL 'DKNPDSIO'               USING PDS-DDB-PARM-LIST
                                        I-O-AREA.

6000-CLOSE-DDB.

    SET  PDS-DDB-CLOSE            TO TRUE.

    CALL 'DKNPDSIO'               USING PDS-DDB-PARM-LIST
                                        I-O-AREA.

7000-PROCESS-PDSIO-ERROR.

    ...................
    ...................
    ...................
```

Figure 4-7 (Part 3 of 3). DKNPDSIO Subroutine Example 1

Example 2

Operation:

Process member MEMBER1, on the PDS allocated to ddname DDA, until a a record with a “T” in its first byte is read.

Assembler Code:

```
        .................
        .................
        .................
*
        MVC   USDDNAME,DDNAMEA      SET PDS DDNAME
        MVC   USMEMBER,MEMBER1      SET MEMBER NAME
        MVC   USREQ,OPENIN          SET OPEN REQUEST
        LA    R1,PARMS              GET PDS IO SERVICES PARM LIST @
```

Figure 4-8 (Part 1 of 3). DKNPDSIO Subroutine Example 2

```
         L     R15,PDSIO@           GET PDS IO SERVICES @
         BASSM R14,R15              CALL PDS IO TO OPEN PDS
         CLC   USSTAT,REQDONE       REQUEST COMPLETED ?
         BNE   PDSIOER              N. PROCESS ERROR
*
READREC  MVC   USREQ,READ           SET READ RECORD REQUEST
         LA    R1,PARMS             GET PDS IO SERVICES PARM LIST @
         L     R15,PDSIO@           GET PDS IO SERVICES @
         BASSM R14,R15              CALL PDS IO TO READ RECORD
         CLC   USSTAT,EOF           ALL RECORDS PROCESSED ?
         BE    CONTINUE             Y. CONTINUE PROCESSING
         CLC   USSTAT,REQDONE       RECORD READ ?
         BNE   PDSIOER              N. PROCESS ERROR
         CLC   IOCODE,MARKDREC      MARKED RECORD ?
         BNE   READREC              N. PROCESS NEXT RECORD
*
         MVC   USREQ,CLOSE          SET CLOSE REQUEST
         LA    R1,PARMS             GET PDS IO SERVICES PARM LIST @
         L     R15,PDSIO@           GET PDS IO SERVICES @
         BASSM R14,R15              CALL PDS IO TO CLOSE PDS
         CLC   USSTAT,REQDONE       REQUEST COMPLETED ?
         BE    CONTINUE             Y. CONTINUE PROCESSING
*
PDSIOER  DS   0H                    PDS IO ERROR ROUTINE
         ..............
         ..............
         ..............
*
CONTINUE DS   0H                    CONTINUE PROCESSING
         ..............
         ..............
         ..............
*
MARKDREC DC    C'T'                 1ST 'DO NOT PROCESSED' CODE
DDNAMEA  DC    CL8'DDA'             PDS DDNAME
MEMBER1  DC    CL8'MEMBER1'         MEMBER NAME
PDSIO@   DC    V(DKNPDSIO)          PDS IO SERVICES @
*
IOAREA   DS   0XL80                 I/O AREA
IOCODE   DS    C                    . RECORD CODE
IODATA   DS    XL79                 . DATA AREA
*
         DS   0H                    REQUEST CODES
OPENIN   DC    C'OI'                . OPEN INPUT
CLOSE    DC    C'C '                . CLOSE
READ     DC    C'R '                . READ RECORD
*
         DS   0H                    RETURN CODES
REQDONE  DC    H'0'                 . REQUEST COMPLETED
EOF      DC    H'16'                . END-OF-FILE
*
PARMS    DS   0H                    DKNPDSIO PARM LIST
         DC    A(USERAREA)          . CALL PARMS @
         DC    A(IOAREA)            . I/O AREA @
*
```

Figure 4-8 (Part 2 of 3). DKNPDSIO Subroutine Example 2

```
         COPY  DKNPDSA               PDS IO SERVICES CALL PARAMETERS
         ..................
         ..................
         ..................
```

Figure 4-8 (Part 3 of 3). DKNPDSIO Subroutine Example 2

DKNQGET — Perform QSAM Input Processing

DKNQGET may be called to read both permanently allocated data sets and dynamically allocated temporary data sets.

Input data sets do not need to be defined (no assembler DCBs or COBOL FDs are required) in the programs using the DKNQGET services.

No special calls are required to open input data sets. They get opened on the first read call.

A special call is required to close permanently allocated data sets, only when they are not read to end-of-file. Input data sets are automatically closed when the end-of-file condition is reached.

A special call is always required to deallocate dynamically allocated data sets.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 4-8. DKNQGET Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNQGET Call Parameters	DKNQGETC	DKNQGETA
Record I/O Area		
Application Task Control Block	DKNCATCB	DKNAPTCB

DKNQGET may return the following completion codes in the call parameters:

Table 4-9. DKNQGET Return Codes

Code (dec)	Description
+00	Request completed.
+04	Invalid request code.
+08	Memory allocation error.
+12	Dynamic data set allocation services load error.
+16	Open error.
+20	Read error.
+24	Close error.
+28	Dynamic data set allocation services delete error.
+32	End-of-file condition detected.

Restrictions

Before the first call to DKNQGET, the RESERVED-AREA-1 and RESERVED-AREA-2 fields, in the DKNQGET call parameters, must be initialized to binary zeroes.

After the first call to DKNQGET, only the REQUEST field may be changed in the call parameters.

Dynamically allocated temporary input data sets must have been allocated by either DKNTDYNA or DKNQPUT. Both the original ddname and the DKNQPUT returned changed ddname need to be specified in the DKNQGET call parameters.

Example 1

Operation:

Sequentially process the records contained in a QSAM data set permanently allocated to ddname DDA. Stop processing if a record has a code of “S” in its first byte.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSQGET.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  DDNAME-A                      PIC X(08) VALUE 'DDA'.

 01  I-O-AREA.
     05  IO-RECORD-CODE            PIC X(01).
         88  IO-STOP                         VALUE 'S'.
     05  FILLER                    PIC X(79).

 COPY DKNQGETC                      REPLACING ==:TAG:==
                                    BY        ==DDA==.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  TCBSPARM                      PIC X(68).
/
 PROCEDURE DIVISION                 USING APTCB
                                          TCBSPARM.

     SET  QGET-DDA-GET              TO TRUE.
     MOVE DDNAME-A                  TO QGET-DDA-ORIG-DDNAME.
```

Figure 4-9 (Part 1 of 2). DKNQGET Subroutine Example 1

```
      PERFORM 1000-PROCESS-DDA
          UNTIL IO-STOP
            OR  NOT QGET-DDA-GOOD.

      IF  QGET-DDA-GOOD
          PERFORM 2000-CLOSE-DDA
          IF  NOT QGET-DDA-GOOD
              PERFORM 3000-PROCESS-QGET-ERROR
          END-IF
      ELSE
          IF  NOT QGET-DDA-END-OF-FILE
              PERFORM 3000-PROCESS-QGET-ERROR.

      GOBACK.

  1000-PROCESS-DDA.

      CALL 'DKNQGET'                 USING QGET-DDA-PARM-LIST
                                           I-O-AREA
                                           APTCB.


      ...................
      ...................
      ...................

  2000-CLOSE-DDA.

      SET  QGET-DDA-CLOSE            TO TRUE.

      CALL 'DKNQGET'                 USING QGET-DDA-PARM-LIST
                                           I-O-AREA
                                           APTCB.

  3000-PROCESS-QGET-ERROR.

      ...................
      ...................
      ...................
```

Figure 4-9 (Part 2 of 2). DKNQGET Subroutine Example 1

Example 2

Operation:

Sequentially process the records contained in a dynamically allocated temporary data set. The data set original and changed ddnames are contained in fields ORDDN and CHDDN, respectively.

Assembler Code:

```
TSQGET   CSECT
TSQGET   AMODE 31
TSQGET   RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
         L     R4,4(R1)             GET DDNAMES @
         USING DDNAMES,R4           BASE DDNAMES CONTROL BLOCK
*
         MVC   QGODDN,ORDDN         SET ORIGINAL DDNAME
         MVC   QGCDDN,CHDDN         SET CHANGED DDNAME
         XC    QGRSV1,QGRSV1        INITIALIZE RESERVED AREA 1
         XC    QGRSV2,QGRSV2        INITIALIZE RESERVED AREA 2
*
READREC  MVC   QGREQ,GETTEMP        SET 'GET TEMPORARY' REQUEST
         LA    R1,PARMS             GET QSAM GET PARM LIST @
         L     R15,QGET@            GET QSAM GET @
         BASSM R14,R15              CALL QSAM GET TO READ A RECORD
         CLC   QGSTAT,EOF           ALL RECORDS PROCESSED ?
         BE    DEALLDS              Y. DEALLOCATE DATA SET
         CLC   QGSTAT,REQDONE       RECORD READ ?
         BE    READREC              Y. PROCESS NEXT RECORD
         B     QGETER               N. PROCESS ERROR
*
DEALLDS  MVC   QGREQ,DEALLOC        SET DEALLOCATE REQUEST
         LA    R1,PARMS             GET QSAM GET PARM LIST @
         L     R15,QGET@            GET QSAM GET @
         BASSM R14,R15              CALL QSAM GET TO CLOSE DATA SET
         CLC   QGSTAT,REQDONE       REQUEST COMPLETED ?
         BE    RETURN               Y. EXIT PROGRAM
*
QGETER   DS   0H                    I/O ERROR ROUTINE
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
QGET@    DC    V(DKNQGET)           QSAM GET SERVICES @
*
IOAREA   DS    XL80                 I/O AREA
*
```

Figure 4-10 (Part 1 of 2). DKNQGET Subroutine Example 2

```
         DS   0H                    REQUEST CODES
GETTEMP  DC    C'GT'                . GET TEMPORARY
DEALLOC  DC    C'CD'                . DEALLOCATE
*
         DS   0H                    RETURN CODES
REQDONE  DC    H'0'                 . REQUEST COMPLETED
EOF      DC    H'32'                . END-OF-FILE
*
PARMS    DS   0F                    DKNQGET PARM LIST
         DC    A(QGPLIST)           . CALL PARMS @
         DC    A(IOAREA)            . I/O AREA @
APTCB@   DS    A                    . APTCB @
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         COPY  DKNQGETA             QSAM GET CALL PARAMETERS
*
DDNAMES  DSECT                      ALLOCATED DATA SET DDNAMES
ORDDN    DS    CL8                  . QSAM DATA SET ORIGINAL DDNAME
CHDDN    DS    CL8                  . QSAM DATA SET CHANGED DDNAME
*
         END
```

Figure 4-10 (Part 2 of 2). DKNQGET Subroutine Example 2

DKNQPUT — Perform QSAM Output Processing

DKNQPUT may be called to write to both permanently allocated data sets and dynamically allocated temporary data sets.

Output data sets do not need to be defined (no assembler DCBs or COBOL FDs are required) in the programs using the DKNQPUT services.

No special calls are required to allocate output data sets. They get allocated on the first write call.

No special calls are required to open output data sets. They get opened on the first write call.

A special call is required to close, or close and deallocate, output data sets after the last record has been written.

When creating dynamically allocated temporary data sets, the call parameter "original ddname" is used to access the data set information in the DKNDSAT table. The actual allocated ddname (Thhmmsst, where hhmmsst is the allocation time) is returned in the "changed ddname" field. The allocation DSName is composed of the DKNDSAT DSName and the hmmsst DSName extension.

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 4-10. DKNQPUT Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
DKNQPUT Call Parameters	DKNQPUTC	DKNQPUTA
Record I/O Area		
Application Task Control Block	DKNCATCB	DKNAPTCB

DKNQPUT may return the following completion codes in the call parameters:

Table 4-11. DKNQPUT Return Codes

Code (dec)	Description
+00	Request completed.
+04	Invalid request code.
+08	Memory allocation error.
+12	Dynamic data set allocation services load error.
+16	Open error.
+20	Write error.
+24	Close error.
+28	Dynamic data set allocation error.

Restrictions

Before the first call to DKNQPUT, the RESERVED-AREA-1 and RESERVED-AREA-2 fields, in the DKNQPUT call parameters, must be initialized to binary zeroes.

After the first call to DKNQPUT, only the REQUEST field may be changed in the call parameters.

Example 1

Operation:

Call DKNQPUT to create a dynamically allocated temporary data set.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSQPUT.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  LAST-RECORD-SW                PIC X(01) VALUE 'N'.
     88  LAST-RECORD-WRITTEN                 VALUE 'Y'.

 01  ORIGINAL-DDNAME               PIC X(08) VALUE 'DDA'.

 01  I-O-AREA                      PIC X(80).

 COPY DKNQPUTC                      REPLACING ==:TAG:==
                                    BY        ==DDA==.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  TCBSPARM                      PIC X(68).
/
 PROCEDURE DIVISION                 USING APTCB
                                          TCBSPARM.

     SET  QPUT-DDA-PUT-TEMP         TO TRUE.
     MOVE ORIGINAL-DDNAME           TO QPUT-DDA-ORIG-DDNAME.

     PERFORM 1000-PROCESS-DDA.
         UNTIL LAST-RECORD-WRITTEN
           OR  NOT QPUT-DDA-GOOD.

     IF  QPUT-DDA-GOOD
         PERFORM 2000-CLOSE-DDA.
```

Figure 4-11 (Part 1 of 2). DKNQPUT Subroutine Example 1

```
     IF  NOT QPUT-DDA-GOOD
         PERFORM 3000-PROCESS-QPUT-ERROR.

     GOBACK.

 1000-PROCESS-DDA.

     ...................
     ...................
     ...................

     CALL 'DKNQPUT'                USING QPUT-DDA-PARM-LIST
                                         I-O-AREA
                                         APTCB.

 2000-CLOSE-DDA.

     SET  QPUT-DDA-CLOSE           TO TRUE.

     CALL 'DKNQPUT'                USING QPUT-DDA-PARM-LIST
                                         I-O-AREA
                                         APTCB.

 3000-PROCESS-QPUT-ERROR.

     ...................
     ...................
     ...................
```

Figure 4-11 (Part 2 of 2). DKNQPUT Subroutine Example 1

Example 2

Operation:

Call DKNQPUT to update a data set permanently allocated to ddname DDA.

Assembler Code:

```
TSQPUT   CSECT
TSQPUT   AMODE 31
TSQPUT   RMODE ANY
*
         SAVE  (14,12)             SAVE REGISTERS
         BASR  R12,0               GET ROUTINE @
         USING *,R12               BASE SUBROUTINE
*
         LA    R14,SAVEAREA        POINT TO OUR SAVE AREA
         ST    R14,8(R13)          STORE FORWARD SAVE AREA @
         ST    R13,4(R14)          STORE BACKWARD SAVE AREA @
         LR    R13,R14             POINT TO NEW SAVE AREA
*
```

Figure 4-12 (Part 1 of 3). DKNQPUT Subroutine Example 2

```
         MVC   APTCB@,0(R1)          GET APTCB @
*
         MVC   QPODDN,DDN            SET ORIGINAL DDNAME
         XC    QPRSV1,QPRSV1         INITIALIZE RESERVED AREA 1
         XC    QPRSV2,QPRSV2         INITIALIZE RESERVED AREA 2
*
PROCSREC DS   0H
         ..............
         ..............
         ..............
         B     CLOSEDS               LAST RECORD WRITTEN; CLOSE FILE
*
         ..............
         ..............
         ..............
*
         MVC   QPREQ,PUT             SET 'WRITE' REQUEST
         LA    R1,PARMS              GET QSAM PUT PARM LIST @
         L     R15,QPUT@             GET QSAM PUT @
         BASSM R14,R15               CALL QSAM PUT TO WRITE A RECORD
         CLC   QGSTAT,REQDONE        RECORD WRITTEN ?
         BE    PROCSREC              Y. PROCESS NEXT RECORD
         B     QPUTER                N. PROCESS ERROR
*
CLOSEDS  MVC   QPREQ,CLOSE           SET 'CLOSE' REQUEST
         LA    R1,PARMS              GET QSAM PUT PARM LIST @
         L     R15,QPUT@             GET QSAM PUT @
         BASSM R14,R15               CALL QSAM PUT TO CLOSE DATA SET
         CLC   QPSTAT,REQDONE        REQUEST COMPLETED ?
         BE    RETURN                Y. EXIT PROGRAM
*
QPUTER   DS   0H                     I/O ERROR ROUTINE
         ..............
         ..............
         ..............
*
RETURN   L     R13,SAVEAREA+4        RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)    RETURN
*
         DKNREGS                     REGISTER EQUATES
*
DDN      DC    CL8'DDA'              QSAM DATA SET DDNAME
QPUT@    DC    V(DKNQPUT)            QSAM PUT SERVICES @
*
IOAREA   DS    XL80                  I/O AREA
*
         DS   0H                     REQUEST CODES
PUT      DC    C'P '                 . PUT
CLOSE    DC    C'C '                 . CLOSE
*
         DS   0H                     RETURN CODES
REQDONE  DC    H'0'                  . REQUEST COMPLETED
*
```

Figure 4-12 (Part 2 of 3). DKNQPUT Subroutine Example 2

```
PARMS    DS   0F                    DKNQPUT PARM LIST
         DC    A(QPPLIST)           . CALL PARMS @
         DC    A(IOAREA)            . I/O AREA @
APTCB@   DS    A                    . APTCB @
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         COPY  DKNQPUTA              QSAM PUT CALL PARAMETERS
*
         END
```

Figure 4-12 (Part 3 of 3). DKNQPUT Subroutine Example 2

DKNTDYNA — Dynamically Allocate/Unallocate Temporary Data Sets

DKNTDYNA may be called to dynamically allocate and unallocate unique temporary data sets.

Linkage

The standard MVS/ESA linkage conventions are followed.

The following interface control blocks are used to invoke DKNTDYNA:

Table 4-12. DKNTDYNA Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB
Request Code • This is a three character field. Valid codes are: **ALC** Allocate data set. **UNA** Unallocate data set.		
The Temporary Data Set's FD (COBOL) / DCB (assembler)		
ddname Return Area • This is an eight byte field where DKNTDYNA returns the original ddname associated to the data set on the allocation call. That ddname must be passed to DKNTDYNA on the unallocation call, to have the data set's ddname restored to its original value.		

DKNTDYNA may return the same completion codes as DKNDYNA in register 15 (assembler programs) or the RETURN-CODE special register (COBOL programs). See "DKNDYNA — Dynamically Allocate/Unallocate Permanent Data Sets" on page 4-11 for a description of the DKNDYNA return codes.

A supervisor message, containing the ddname being allocated, is issued by DKNTDYNA whenever it does not return a successful completion return code.

Restrictions

To use this feature, entries must be defined in the DKNDSAT table for the data sets being allocated. See "Dynamically Allocating Data Sets" in the *CPCS-I Customization Guide* for information on how to create DKNDSAT table entries.

Allocated data sets have a DSName equal to the DSName specified in the DKNDSAT table entry, with the extension Thhmmsst, where hhmmsst is the allocation time (hour, minutes, seconds and tenth of seconds). The allocated ddnames are named Thhmmsst, which replace the application program and DKNDSAT table ddnames during the task's execution until the data sets are unallocated.

Example 1

Operation:

Allocate a temporary data set with a record length of 80 bytes, a primary space of 100 tracks, and a secondary space of 100 tracks, in disk unit SYSDA. It is assumed that the DKNDSAT table contains an entry with the parameters DYNDEV=DISK, DYNDDN=DDNAME1, DYNDSN=CPCS.DSNAM1, DYNSTAT=NEW, DYNNORM=DELETE, DYNCOND=DELETE, DYNUNIT=SYSDA, DYNSTYP=TRK, DYNPSPA=100, DYNSSPA=100, DYNLRCL=80, and DYNBSIZ=800. As a result of the DKNTDYNA call, data set "CPCS.DSNAME1.Thhmmsst" will be allocated to ddname Thhmmsst, where hhmmsst is the allocation time.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSTDYNA.
/
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

 FILE-CONTROL.

     SELECT TEMPORARY-DATA-SET ASSIGN DA-S-DDNAME1.
/
 DATA DIVISION.
 FILE SECTION.

 FD  TEMPORARY-DATA-SET
     BLOCK CONTAINS             0
     RECORDING MODE             F
     LABEL RECORDS              STANDARD.

 01  TEMPORARY-DATA-SET-RECORD  PIC X(80).
/
 WORKING-STORAGE SECTION.

 01  TDYNA-PARMS.
     05  TP-REQUEST-CODE        PIC X(03).
         88  TP-ALLOCATE                   VALUE 'ALC'.
         88  TP-UNALLOCATE                 VALUE 'UNA'.
     05  TP-ORIGINAL-DDNAME     PIC X(08).

 01  SUCCESSFUL-OPERATION       PIC 9(01)  VALUE 0.
/
 LINKAGE SECTION.

 COPY DKNCATCB.

 01  START-PARMS                PIC X(68).
/
 PROCEDURE DIVISION             USING APTCB
                                      START-PARMS.
```

Figure 4-13 (Part 1 of 2). DKNTDYNA COBOL Example 1

```
      PERFORM 1000-ALLOCATE-DATA-SET.

      IF  RETURN-CODE            = SUCCESSFUL-OPERATION
          PERFORM 2000-PROCESS-DATA-SET
          PERFORM 3000-UNALLOCATE-DATA-SET.

      GOBACK.

  1000-ALLOCATE-DATA-SET.

      SET TP-ALLOCATE            TO TRUE.

      CALL 'DKNTDYNA'            USING APTCB
                                       TP-REQUEST-CODE
                                       TEMPORARY-DATA-SET
                                       TP-ORIGINAL-DDNAME.

  2000-PROCESS-DATA-SET.

      ...................
      ...................
      ...................

  3000-UNALLOCATE-DATA-SET.

      SET TP-UNALLOCATE          TO TRUE.

      CALL 'DKNTDYNA'            USING APTCB
                                       TP-REQUEST-CODE
                                       TEMPORARY-DATA-SET
                                       TP-ORIGINAL-DDNAME.
```

Figure 4-13 (Part 2 of 2). DKNTDYNA COBOL Example 1

Assembler Code:

```
TSTDYNA  CSECT
TSTDYNA  AMODE 24
TSTDYNA  RMODE 24
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
         MVC   DPREQCDE,ALLO        SET 'ALLOCATE' REQUEST CODE
         OI    PARMSEND,ENDLIST     SET 'END OF LIST' FLAG
         LA    R1,PARMS             GET DKNTDYNA PARM LIST @
         L     R15,TDYNA@           GET DKNTDYNA @
```

Figure 4-14 (Part 1 of 2). DKNTDYNA Assembler Example 1

```
         BASSM R14,R15              CALL DKNTDYNA TO ALLOCATE
         LTR   R15,R15              SUCCESSFUL OPERATION ?
         BNZ   TDYNAERR             N. PROCESS DKNTDYNA ERROR
         ................           DATA SET'S OPEN, WRITES & CLOSE
         ................
         ................
*
         MVC   DPREQCDE,UNALLO      SET 'UNALLOCATE' REQUEST CODE
         OI    PARMSEND,ENDLIST     SET 'END OF LIST' FLAG
         LA    R1,PARMS             GET DKNTDYNA PARM LIST @
         L     R15,TDYNA@           GET DKNTDYNA @
         BASSM R14,R15              CALL DKNTDYNA TO ALLOCATE
         LTR   R15,R15              SUCCESSFUL OPERATION ?
         BE    RETURN               Y. EXIT PROGRAM
*
TDYNAERR DS   0H                    PROCESS DKNTDYNA ERROR
         ................
         ................
         ................
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
TDYNA@   DC    V(DKNTDYNA)          DKNDYNA @
*
ENDLIST  EQU   X'80'                'END OF LIST' FLAG
*
PARMS    DS   0F                    DKNTDYNA PARM LIST
APTCB@   DS    A                   . APTCB @
         DC    A(DPREQCDE)         . REQUEST CODE FIELD @
         DC    A(DCB1)             . DCB @
PARMSEND DC    A(DDNAMESV)         . ORIGINAL DDNAME SAVE AREA @
*
DPREQCDE DS    CL3                 REQUEST CODE
ALLO     DC    C'ALC'              . ALLOCATE DATA SET
UNALLO   DC    C'UNA'              . UNALLOCATE DATA SET
*
DDNAMESV DS    CL8                 ORIGINAL DDNAME SAVE AREA
*
SAVEAREA DS  18F                   OUR SAVE AREA
*
DCB1     DCB   DSORG=PS,DDNAME=DDNAME1,MACRF=PM
*
         END
```

Figure 4-14 (Part 2 of 2). DKNTDYNA Assembler Example 1

DKNVSMIO — Perform VSAM I/O Processing

DKNVSMIO provides a high-level program interface to VSAM data sets and reduces application-code repetition in VSAM data set manipulation. DKNVSMIO allows:

- Interfacing to CPCS-I VSAM Manager to permit VSAM data sets to be treated as CPCS-I resources.
- Serializing record access to ensure data integrity.
- Providing access sequentially or directly.
- Reading, writing, deleting and updating records.
- Retrieving the DSName for known ddnames.

Accessed data sets do not need to be defined (no assembler DCBs or COBOL FDs are required) in the programs using the DKNVSMIO services.

Notes:

1. See “CPCS-I VSAM Manager” in the *CPCS-I Programming Guide*, for a description of the CPCS-I VSAM Manager functions.
2. A VSAM data set is defined as a CPCS-I resource when it is listed in the CPCS-I VSAM table. See “Accessing the VSAM Table” in the *CPCS-I Customization Guide* for information on how to define entries in the CPCS-I VSAM table.
3. VSAM data sets may be accessed by DKNVSMIO even if they are not defined as CPCS-I resources. However, having VSAM data sets defined as CPCS-I resources eliminates overhead, because the data sets are closed only once, at CPCS-I shutdown time (a “close” request results in just freeing temporary buffers).

Linkage

The standard MVS/ESA linkage conventions are followed.

Information is passed in the following interface control blocks:

Table 4-13 (Page 1 of 2). DKNVSMIO Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
Application Task Control Block	DKNCATCB	DKNAPTCB

Table 4-13 (Page 2 of 2). DKNVSMIO Communication Control Blocks

Control Block	COBOL Copybook	Assembler Macro/ Copybook
VSAM Open File List • The data set ID is specified on "open" requests for data sets listed in the CPCS-I VSAM table. The field should contain blanks if it is not defined as a CPCS-I resource. • The ddname is specified on "open" requests for data sets not listed in the CPCS-I VSAM table. • The operation code is always specified. • The access type is specified for read, write and delete operations. • The data buffer address and length are specified for each read or write operation. • The key address is specified for keyed and generic keyed accesses. • The generic key length is specified for generic keyed reads. • Successful "open" operations return information in the ddname, DSName, maximum record key length, key offset, maximum record length, data set and record organization, ACB address and length, and CPCS-I VSAM table entry address fields, and the RPL area initialized as a VSAM request parameter list. • Successful "read" operations update the record length, and last key (KSDS data sets) or last RBA (ESDS data sets) fields, and returns the record read in the area pointed to by the buffer pointer field.	DKNCRVSM	VSMVOFL

DKNVSMIO returns both VSAM and its own completion codes in the RETURN-CODE special register (COBOL programs) or register 15 (assembler programs). The DKNVSMIO return codes, documented in assembler copybook VSGENERR, are returned as negative values in order to differentiate them from the VSAM codes. In addition, the VSAM reason code is returned in the REASON field of the VSAM open file list control block. A character string identifying the VSAM operation being performed, and containing the VSAM return and reason codes, is also returned in the PROBAID field (the message format is `OP=xxxxxx RC=Xyy REASON=Xzz`) of the VSAM open file list control block. VSAM return codes are documented in the MVS/ESA library.

DKNVSMIO may return the following non-VSAM completion codes:

Table 4-14 (Page 1 of 2). DKNVSMIO non-VSAM Return Codes

Code (dec)	Description
+00	Request completed.
-04	Not enough virtual storage available.
-10	Initialization error.

Table 4-14 (Page 2 of 2). DKNVSMIO non-VSAM Return Codes

Code (dec)	Description
-11	Exclusive VOFL base use not available.
-12	Deleted record read from ESDS data set.
-13	Access requested for ESDS untouchable record.
-14	Invalid operation code.
-15	Exclusive record use not available.
-16	Exclusive data set use not available.
-17	Load failed for a lower-level module.
-18	Invalid data set name buffer address.
-19	Invalid ddname.
-20	Invalid parameter-list address. This can occur because of a parameter list address of zero or because the parameter list is not ended (the high-order bit of the last address is not on).
-21	Data set not found for specified ddname.
-22	Information retrieval request failed.
-23	Invalid number of VSMOFLs requested.
-24	Data set ID not in CPCS-I VSAM table.
-25	Data set cannot be opened for being forced offline.
-26	Invalid ESDS RBA in update request (caller should ensure that the record to be updated has been read for update).
-27	Record organization not supported.
-28	ACB cannot be generated by VSAM.
-29	Write requested for “read only” data set ID.
-30	Open error on an empty or re-used data set.
-31	“Read only” access requested for an empty data set or data set to be emptied on next open (to be initialized, a write must occur).
-32	CPCS-I VSAM Manager abended during an open or close operation.
-33	CPCS-I VSAM Manager received an invalid request.

Note: Instead of calling DKNVSMIO, programs may call subroutine DKNVSCOB, a DKNVSMIO interface that formats return codes and does not require pointers in the COBOL communication control block. Macro DKNVSCP may be used by assembler programs to reference the DKNVSCOB communication control block.

Restrictions

Only KSDS and ESDS record organizations are supported.

Direct access is only supported for KSDS record organizations.

The communication control block must be placed in the Linkage Section (COBOL programs) because of some fields that are defined as pointers.

ESDS records must be prefixed by the ESDS record header. Macro VSESDS may be used to reference this area in assembler programs. Copybook DKNCVHDR may be used to reference it in COBOL programs. The header information is used to support “deleted” (ESDS record deletions are not allowed by VSAM), and “untouchable” record flags.

Example 1

Operation:

Call DKNVSMIO to update a KSDS VSAM data set that is defined as a non read-only CPCS-I resource with an ID of DSA. The data set is processed as follows:

1. Key range 100100–111300 records are read sequentially forward.
2. Code “D” records are deleted.
3. Code “U” records are updated, assigning them a code of “N”.
4. A blank-data record is inserted following each “I” code record. The inserted record’s key is the preceding record’s key incremented by 1.

COBOL Code:

```
 ID DIVISION.
 PROGRAM-ID. TSVSMIO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
/
 WORKING-STORAGE SECTION.

 01  DSP-RETURN-CODE               PIC S9(04).

 01  SUFFIX-INCREMENT              PIC 9(01) VALUE 1.

 01  DATA-SET-ID                   PIC X(08) VALUE 'DSA'.
 01  I-O-AREA-LENGTH               PIC 9(08) VALUE 80.

 01  START-KEY                     PIC X(06) VALUE '100100'.
 01  END-KEY                       PIC X(06) VALUE '111300'.

 01  INSERT-RECORD.
     05  IR-KEY.
         10  FILLER                PIC X(04).
         10  IRK-SUFFIX            PIC 9(02).
     05  FILLER                    PIC X(74) VALUE ' '.

 01  SUCCESSFUL-OPERATION          PIC 9(02).
/
 LINKAGE SECTION.
```

Figure 4-15 (Part 1 of 4). DKNVSMIO Subroutine Example 1

```
 COPY DKNCATCB.

 COPY DKNCRVSM.

     03  I-O-AREA.
         05  IO-KEY                PIC X(06).
         05  IO-CODE               PIC X(01).
             88  IOC-UPDATE                 VALUE 'U'.
             88  IOC-INSERT                 VALUE 'I'.
             88  IOC-NEW                    VALUE 'N'.
             88  IOC-DELETE                 VALUE 'D'.
         05  FILLER                PIC X(73).

     03  LOCATE-KEY                PIC X(06).

 01  TCBSPARM                      PIC X(68).
/
 PROCEDURE DIVISION                 USING APTCB
                                          TCBSPARM.

     PERFORM 1000-OPEN-DATA-SET.

     IF  RETURN-CODE                = SUCCESSFUL-OPERATION
         PERFORM 2000-READ-START-RECORD

         IF  RETURN-CODE            = SUCCESSFUL-OPERATION
             PERFORM 3000-PROCESS-RECORDS
                 UNTIL IO-KEY       > END-KEY
                 OR    RETURN-CODE  NOT = SUCCESSFUL-OPERATION

             IF RETURN-CODE         = SUCCESSFUL-OPERATION
                 PERFORM 4000-CLOSE-DATA-SET.

     IF  RETURN-CODE                NOT = SUCCESSFUL-OPERATION
         PERFORM 5000-PROCESS-RETURN-CODE.

     GOBACK.

 1000-OPEN-DATA-SET.

     MOVE DATA-SET-ID               TO VSM-VO-FILE-ID.
     SET VSMOP-OPEN-DYN             TO TRUE.

     CALL 'DKNVSMIO'                USING APTCB
                                          VSM-VO-FILE-ALIAS.

 2000-READ-START-RECORD.

     SET VSMOP-READ                 TO TRUE.
     SET VSMAC-KEYED                TO TRUE.
     MOVE START-KEY                 TO LOCATE-KEY.
     SET VSM-VO-KEY-ADDRS           TO ADDRESS OF LOCATE-KEY.
     MOVE I-O-AREA-LENGTH           TO VSM-VO-BUFR-LEN.
     SET VSM-VO-BUFR-ADDRS          TO ADDRESS OF I-O-AREA.
```

Figure 4-15 (Part 2 of 4). DKNVSMIO Subroutine Example 1

```
    CALL 'DKNVSMIO'               USING APTCB
                                        VSM-VO-FILE-ALIAS.

3000-PROCESS-RECORDS.

    IF  IOC-DELETE
        PERFORM 3300-DELETE-RECORD
    ELSE
        IF  IOC-UPDATE
            PERFORM 3500-UPDATE-RECORD
        ELSE
            IF  IOC-INSERT
                PERFORM 3700-INSERT-RECORD.

    IF  RETURN-CODE               = SUCCESSFUL-OPERATION
        PERFORM 3900-READ-NEXT-RECORD.

3300-DELETE-RECORD.

    SET VSMOP-DEL                 TO TRUE.
    MOVE IO-KEY                   TO LOCATE-KEY.
    SET VSM-VO-KEY-ADDRS          TO ADDRESS OF LOCATE-KEY.

    CALL 'DKNVSMIO'               USING APTCB
                                        VSM-VO-FILE-ALIAS.

3500-UPDATE-RECORD.

    SET IOC-NEW                   TO TRUE.
    SET VSMOP-WRITE               TO TRUE.
    SET VSMAC-UPDATE              TO TRUE.

    CALL 'DKNVSMIO'               USING APTCB
                                        VSM-VO-FILE-ALIAS.

3700-INSERT-RECORD.

    MOVE IO-KEY                   TO IR-KEY.
    ADD SUFFIX-INCREMENT          TO IRK-SUFFIX.
    MOVE INSERT-RECORD            TO I-O-AREA.
    SET VSMOP-WRITE               TO TRUE.

    CALL 'DKNVSMIO'               USING APTCB
                                        VSM-VO-FILE-ALIAS.

3900-READ-NEXT-RECORD.

    SET VSMOP-READ                TO TRUE.
    SET VSMAC-SEQ                 TO TRUE.

    CALL 'DKNVSMIO'               USING APTCB
                                        VSM-VO-FILE-ALIAS.

4000-CLOSE-DATA-SET.
```

Figure 4-15 (Part 3 of 4). DKNVSMIO Subroutine Example 1

```
     SET VSMOP-CLOSE               TO TRUE.

     CALL 'DKNVSMIO'               USING APTCB
                                         VSM-VO-FILE-ALIAS.

 5000-PROCESS-RETURN-CODE.

     ...................
     ...................
     ...................
```

Figure 4-15 (Part 4 of 4). DKNVSMIO Subroutine Example 1

Example 2

Operation:

Call DKNVSMIO to update an ESDS VSAM data set that is not defined as a CPCS-I resource and is allocated to the ddname DSA. The data set is processed sequentially forward, as follows:

1. Code “D” records are deleted.
2. Code “U” records are updated, assigning them a code of “N”.

Assembler Code:

```
TSVSMIO  CSECT
TSVSMIO  AMODE 31
TSVSMIO  RMODE ANY
*
         SAVE  (14,12)              SAVE REGISTERS
         BASR  R12,0                GET ROUTINE @
         USING *,R12                BASE SUBROUTINE
*
         LA    R14,SAVEAREA         POINT TO OUR SAVE AREA
         ST    R14,8(R13)           STORE FORWARD SAVE AREA @
         ST    R13,4(R14)           STORE BACKWARD SAVE AREA @
         LR    R13,R14              POINT TO NEW SAVE AREA
*
         MVC   APTCB@,0(R1)         GET APTCB @
*
         LOAD  EP=DKNVSMIO,ERRET=LOADERR  LOAD VSAM SERVICES MODULE
         ST    R0,VSMIO@            SAVE VSAM SERVICES ENTRY POINT
*
         MVC   VSMDDN,DSDDN         SET DATA SET DDNAME
         MVI   VSMOPCDE,VSMOP_OPENW  SET 'OPEN READ/WRITE' REQUEST
         OI    VSMVOFL@,ENDLIST     SET 'END OF LIST' flag
         LA    R1,PARMS             GET VSAM SERVICES PARM LIST @
         L     R15,VSMIO@           GET VSAM SERVICES @
```

Figure 4-16 (Part 1 of 3). DKNVSMIO Subroutine Example 2

```
         BASSM R14,R15               CALL VSAM SERVICES
         LTR   R15,R15               SUCCESSFUL OPEN ?
         BNZ   VSMIORC               N. PROCESS RETURN CODE
*
PROCESS  MVI   VSM_ACCESS,VSM_SEQ    SET 'SEQUENTIAL' ACCESS
         MVI   VSMOPCDE,VSMOP_READ   SET 'READ RECORD' REQUEST CODE
         LA    R4,IOAREA             SET I/O
         ST    R4,VSMBUFF_PTR          AREA @
         LA    R6,IOAREALE           SET I/O AREA
         ST    R6,VSMBUFF_LEN          LENGTH
         LA    R1,PARMS              GET VSAM SERVICES PARM LIST @
         L     R15,VSMIO@            GET VSAM SERVICES @
         BASSM R14,R15               CALL VSAM SERVICES
         C     R15,FILEEND           END OF FILE REACHED ?
         BE    CLOSEFL               Y. CLOSE FILE & EXIT PROGRAM
         C     R15,DELREC            'DELETED RECORD' READ ?
         BE    PREND                 Y. SKIP IT
         C     R15,UNTREC            'UNTOUCHABLE RECORD' READ ?
         BE    PREND                 Y. SKIP IT
         LTR   R15,R15               SUCCESSFUL READ ?
         BNZ   VSMIORC               N. PROCESS RETURN CODE
         CLI   IORCODE,IORDEL        'DELETE' RECORD CODE ?
         BNE   UPDCHK                N. CHECK FOR UPDATE RECORD
*
DELETE   MVI   VSMOPCDE,VSMOP_DELET  SET 'DELETE' REQUEST CODE
         LA    R1,PARMS              GET VSAM SERVICES PARM LIST @
         L     R15,VSMIO@            GET VSAM SERVICES @
         BASSM R14,R15               CALL VSAM SERVICES
         LTR   R15,R15               SUCCESSFUL DELETE ?
         BNZ   VSMIORC               N. PROCESS RETURN CODE
*
UPDCHK   CLI   IORCODE,IORUPD        'UPDATE' RECORD CODE ?
         BNE   PREND                 N. RECORD PROCESSED
*
UPDATE   MVI   IORCODE,IORNEW        SET NEW RECORD CODE
         OI    VSM_ACCESS,VSM_UPDATE SET 'SEQUENTIAL + UPDATE' ACCESS
         MVI   VSMOPCDE,VSMOP_WRITE  SET 'WRITE' REQUEST CODE
         LA    R1,PARMS              GET VSAM SERVICES PARM LIST @
         L     R15,VSMIO@            GET VSAM SERVICES @
         BASSM R14,R15               CALL VSAM SERVICES
         LTR   R15,R15               SUCCESSFUL UPDATE ?
         BNZ   VSMIORC               N. PROCESS RETURN CODE
*
PREND    B     PROCESS               PROCESS NEXT RECORD
*
CLOSEFL  MVI   VSMOPCDE,VSMOP_CLOSE  SET 'CLOSE' REQUEST CODE
         LA    R1,PARMS              GET VSAM SERVICES PARM LIST @
         L     R15,VSMIO@            GET VSAM SERVICES @
         BASSM R14,R15               CALL VSAM SERVICES
         LTR   R15,R15               SUCCESSFUL CLOSE ?
         BNZ   VSMIORC               N. PROCESS RETURN CODE
         B     RETURN                EXIT PROGRAM
*
```

Figure 4-16 (Part 2 of 3). DKNVSMIO Subroutine Example 2

```
LOADERR  DS   0H                    PROCESS DKNMVSIO LOAD ERROR
         .................
         .................
         .................
*
VSMIORC  DS   0H                    PROCESS DKNMVSIO RETURN CODE
         .................
         .................
         .................
*
RETURN   L     R13,SAVEAREA+4       RESTORE CALLER'S SAVE AREA @
         XRETURN (14,12),,RC=(15)   RETURN
*
         DKNREGS                    REGISTER EQUATES
*
DSDDN    DC    CL8'DSA'             DATA SET DDNAME
*
         DS   0F                    RETURN CODES
FILEEND  DC    F'8'                 . 'END OF FILE'
DELREC   DC    F'-12'               . 'DELETED RECORD READ'
UNTREC   DC    F'-13'               . 'UNTOUCHABLE RECORD READ'
*
VSMIO@   DS    F                    DKNVSMIO LOAD ADDRESS
*
IOAREA   DS   0XL80                 I/O AREA
         DS    XL12                 . ESDS RECORD HEADER
IORCODE  DS    C                    . RECORD CODE
IORDEL   EQU   C'D'                   - DELETE
IORUPD   EQU   C'U'                   - UPDATE
IORNEW   EQU   C'N'                   - NEW
         DS    XL67                 . DATA
IOAREALE EQU   *-IOAREA             I/O AREA LENGTH
*
PARMS    DS   0F                    DKNVSMIO PARM LIST
APTCB@   DS    A                    . APTCB @
VSMVOFL@ DC    A(VSMVOFL)           . VSMOFL @
*
SAVEAREA DS  18F                    OUR SAVE AREA
*
         VSMVOFL DSECT=NO           VSMVOFL CONTROL BLOCK
*
ENDLIST  EQU   X'80'                 'END OF LIST' FLAG
*
         END
```

Figure 4-16 (Part 3 of 3). DKNVSMIO Subroutine Example 2

Glossary

This glossary defines important terms and abbreviations used in this manual. If you do not find the term you are looking for, refer to the Index or to the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

A

ABA. American Bankers Association.

ABA number. (1) A numbering system devised by the ABA to provide exact identification of financial institutions. The code structure also identifies the Federal Reserve Bank and branch. (2) The MICR-inscribed field on a US document, containing the financial institution identification number.

account number field. An encoded field, on a check or a deposit slip, that indicates the account held by the drawer of the debit or the recipient of the credit.

adjustment. A change to a credit or debit document that adjusts the balance status of a deposit group (or transaction group).

advice. A letter that is sent to a financial institution or customer from whom checks have been received, advising that errors have been detected in the checks or in the listing that accompanied the checks.

ALS. Application Library Services.

American Bankers Association (ABA). Among the functions of this group is the specification of banking industry standards for US check-handling documents and procedures.

amount due field. This field is on some UK credit documents, typically utility payments, indicating the amount that is due for payment. It might or might not be the same as the actual amount field which will be encoded by the presenting bank when the credit is paid in.

amount field. An encoded field on an item that represents the amount of that item.

Application Library Services (ALS). See *ImagePlus HPTS Application Library Services.*

application tasks. Those application tasks that are delivered as part of the base CPCS-I program product or product feature.

application program task control block (APTCB). A CPCS-I area created by the applications task (DKNATASK) for every active subtask in the system. This area contains operating system control blocks that are related to the subtask; it also contains addresses and constants used by the CPCS-I executive programs.

APTCB. Application program task control block.

assist document (AST). A document that accompanies incoming work and that supplies information about the work. A remittance/kill list is an example of an assist document.

AST. Assist document.

automatic restart. The process of restarting (continuing) an interrupted entry without having to find and rebatch any item.

B

balanced M-string. The M-string that has been balanced by a balancing product. The balanced M-string is denoted by the string name *eeee-p-p1-p2-p3*-99-*t-sss*.

balancing. The act of bringing two sets of related figures into agreement (for example, reconciling accumulated-detail totals and input-control totals).

bank control file (BCF). A CPCS-I data set that contains control information for multiple bank processing.

Bank Giro Credit (BGC). A UK credit document that may be paid in only through a clearing bank. It may be encoded in MICR or in a mixture of MICR and OCR, but the format of the codeline is broadly similar to a check.

base CPCS-I application tasks. See *application tasks.*

basic direct access method (BDAM). An access method used to directly retrieve or update particular blocks of a data set on a direct access device.

batch. The lowest required level that has monetary control established by a control document. See also *Docket Control Voucher.*

batch number. The number that uniquely identifies a specific batch of documents.

batch slip. A level of control for balancing items. See also *batch* and *Docket Control Voucher.*

BCF. Bank Control File.

BDAM. Basic direct access method.

BGC. Bank Giro Credit.

block. (1) A prime-pass control level consisting of one or more batches. In CPCS-I, this control level is used to total multiple batches. A block can also represent work from a specific source. (2) A data-processing term used to refer to a series of logical records stored contiguously on external storage devices. (3) To insert control documents in preparation for a prime-pass sorter run. See also *data preparation.*

block slip. A level of control for balancing batches. See also *block.*

branch separators. A UK term for user control documents used to separate work for different branches in on-us output pockets.

buffer. A main storage area used as a data-transfer area for physical records being read or written.

bundle. A bundle is a set of documents grouped together for processing and prefixed, for control purposes, by slips (for example, batch).

C

capture. (1) To read the codeline that is inscribed on a document. (2) To make a digitized image of a document. In the HPTS system, full-item images can be captured by the Image Capture System attached to the document processor or by a low-speed scanner attached to a workstation.

cash letter summary. In the US, a listing that summarizes kill lists by giving monetary totals and item controls for each kill list. In the UK, this is referred to as a DCV Summary.

CDM. Codeline Data Matching.

CDMP. Codeline Data Matching Prime.

CDMR. Codeline Data Matching Rejects.

check. (UK = cheque) A draft drawn on a financial institution and payable on demand on or after the date indicated.

check number. See *serial field* or *reference.*

Check Image Management System Data Base (CIMS Data Base). A program in ImagePlus HPTS Application Library Services that stores, gets, and manages document images.

cheque. UK spelling of “check.”

CIMS. Check Image Management System. See *Check Image Management System Data Base.*

clearing house. An organization, established by financial institutions in the same locality, through which checks and other instruments are exchanged and net balances settled.

codeline data matching (CDM). A method by which a computer system controls items on a detail level by comparing the internal data records from a previous pass with data that it reads on the current pass.

codeline data matching prime (CDMP). The process of performing codeline data matching during a CPCS-I prime pass. Document codeline data is matched against DEFT data transmitted from another bank or a branch of the processing bank. See also *document-based electronic funds transfer.*

codeline data matching rejects (CDMR). The process of performing codeline data matching on CPCS-I prime-pass rejects. Document codeline data is matched against Prime/HSRR codeline data that has been repaired (for example, in OLRR or HPTS key entry).

codeline data record. See *data record.*

cold start. An initiation of the CPCS-I region that causes the deletion of the previous contents of the mass data set and the control data sets.

complete task status. This indicates that this task processed successfully for this UOW. See also *task status.*

complete UOW status. This indicates that all tasks in the task list processed successfully or had a bypass status.

component. A set of modules that performs a major function within a system; for example, a compiler or a master scheduler.

component internal data. All data accessible to any modules within a particular component, but not accessible to any part of the system outside this component.

concurrent kill. Producing remittance/kill lists for kill pockets in an entry before the entire entry is processed. The concurrent kill feature is available only with subset processing.

concurrent processing. A system where the processing of prime capture work through subsequent processes (such as reject handling, rehandle sorting, or remittance printing) begins before completing capture for the whole entry.

control block. A storage area that a computer program uses to hold control information.

control document. An encoded document that contains control information, such as the total of the checks that the document controls, the source of the checks, and a code that describes the level of control.

control slip. See *control document.*

control total. The total value or item count for a group of documents.

copy library. A library that contains statements to be modified by the user, accessed by the assembler instruction copy, and inserted into some of the CPCS-I programs.

correspondent financial institution. A financial institution that carries a deposit balance for, or engages in an exchange of services with, another financial institution.

CPCS-I. Check Processing Control System International MVS/ESA.

credit. The opposite of a debit. Common examples are deposit slips and utility payments.

cross record. See *XREC.*

cutoff. (1) The financial institution's designated point for balancing or releasing work before processing continues. (2) The designated time after which the financial institution cannot accept work for processing.

cycle. (1) A group of work or an identification of a group of work processed completely as a single entity. (2) A convenient grouping of work. A cycle normally contains a variable number of entries.

D

DASD. Direct access storage device.

data preparation. The preparation of documents for processing by a high-speed check-processing system.

data record. The electronic representation of the codeline captured from a check, deposit, debit, credit, or control document. The electronic representation can include additional data to help identify the record.

data space. An area of virtual storage that a program can ask the system to create. The area's size can range from 4K bytes to 2 gigabytes, according to the program's request. Unlike an address space, a data space contains only data. Program code cannot run in a data space. Unlike data in a Hiperspace, data in a data space is directly addressable.

DCV. Docket Control Voucher.

DCV summary. A listing that summarizes all of the kill bundles in a DCV summary report by giving monetary and item controls for each remittance list. See also *cash letter summary.*

DCV summary report. Report listing the group of items to be delivered to an endpoint. Grouping of the items is usually by kill bundle.

debit. A transaction that increases an asset or decreases a liability. In normal check-collection terminology, a check is considered a debit.

deferred printing. The method by which data is processed, transferred to a storage device, and later printed (as opposed to printing during the processing of data).

DEFT. Document-based Electronic Funds Transfer.

DEFT input. Electronically captured data that supports processing of paper documents in a codeline data-matching prime pass.

deleted UOW status. This indicates that the string associated with this UOW is deleted. No more processing can be done for this UOW.

deposit slip. A document that details a deposit. The total of the deposit is encoded on the deposit slip. A deposit is considered a credit.

DFD. Data Flow Diagram.

direct access storage device (DASD). A device in which access time is independent of the location of the data.

distributed string (D-string). The distribution task reads I-strings that the MICR task created and produces D-strings. Each D-string contains the records that correspond to all of the documents in a given pocket of the document processor.

divider slip. A control document that is used to separate kill bundles during machine sorting. It can also be used to support the resynchronization of codeline data matching during subsequent-pass processing.

Docket Control Voucher (DCV). A UK document used to prefix a batch of documents for exchange between clearing operations. A DCV is considered a Batch Slip by CPCS-I. See also *batch.*

document-based electronic funds transfer (DEFT). The transmission, reception, and processing of codeline data sent or received electronically from another

location together with the documents. The data is used in codeline data matching and reconciliation to reduce rejects and balance work.

document processor. A device that can read encoded characters from documents and sort the documents into multiple pockets.

document processor station. A work station consisting of a document processor and a terminal for operator communication.

drawer. The person on whose account a check is being drawn.

D-string. Distributed string.

E

ECDM. Extended codeline data matching.

enclosed and not listed. A condition that exists when an item is in a batch of checks but is not listed on the incoming kill/remittance list or inscriber tape.

encode. To imprint a MICR field on a document. The CPCS-I database contains the information that is encoded. Synonymous with *inscribe*.

encoder. A machine that encodes or inscribes. Synonymous with *inscriber*.

endorsement. (1) The signature of the endorser; (2) the stamp of a financial institution or company.

endorser. (1) A person or financial institution, other than the maker, who presents a check for payment. (2) A device that stamps an endorsement.

endpoint. The destination of an item (debit or credit).

enhanced reject processing. The pockets used in this processing are alternate reject pockets, eligible to receive a reject item and/or an unencoded reject item. These pockets are defined in the J sort pattern definition record with values of J, E, and U respectively.

entry. A variable number of documents that are processed as a single group of work. Normally consists of a number of blocks and batches.

entry number. The number of the first tracer group within an entry.

EPC. Extended process control field.

ERP. Enhanced reject processing.

error description. The detailed description of an error created, detected, and corrected by the processing financial institution.

exception printing. The printing of only the data that requires action external to a computer.

extended codeline data matching (ECDM). A feature available on the 389x/XP Series document processors. It allows the matching criteria to be changed on a per-document basis (based on the perfectly read fields or on the number of digit errors in a field) and increases the chance of a successful match.

extended process control field (EPC). An optional encoded field that indicates special handling (such as return or truncation).

F

fine-sort. (1) The sorting of items, for example, into account number order for filing. (2) The sorting of items for a single account into serial-number order as a customer service.

fine sort group (FSG). A group of documents that have been block-sorted under CPCS-I for fine sorting. Each FSG has a unique CPCS-I endpoint and does not enter fine sorting until all work for that FSG has been processed through all preceding passes.

flip-flop. An event that occurs when the volume to which you are writing a file becomes full. The writing continues on a new volume and the full volume is backed up.

float. The portion of a financial institution's total deposits, or of a depositor's account, that represents items (for example, checks) in the process of collection.

flow code. A 3-digit number (mnemonic) that represents an ordered list of tasks.

flow control. The pairing of a CPCS-I string with a task list through the specification of sort type, pass-pocket history, string type, and flow code.

FSG. Fine sort group.

full-page printing. A method of page formatting in which items are listed in as many columns as can be contained on the page (for example, the first 50 items in column 1, the second 50 in column 2, and so on).

functional unit of work. This unit of work corresponds to a CPCS-I string or subset string.

funds availability. The portion of the financial institution's total deposits or of a depositor's account that represents items (for example, checks) that have been collected and are now available. This includes cash deposited and checks drawn on the depositor's financial institution.

G

generated total. The total value or item count of checks that are processed by the computer.

H

held task status. This indicates that this task should be the next task to process, but a condition external to CPCS-I must complete first. See also *task status.*

High Performance Transaction System (HPTS). See *ImagePlus High Performance Transaction System.*

high-speed reject re-entry. The re-entering into the document processor of reconditioned documents that have previously been sorted to the system reject pocket (pocket 1-1).

Hiperspace. A range of up to two gigabytes of contiguous virtual storage addresses that a program can use as a buffer. Like a data space, a Hiperspace holds only data, not common areas or system data; code does not execute in a Hiperspace. Unlike data in a data space, data in a Hiperspace is not directly addressable.

holdover. (1) Items that were not processed in time to meet their deadline. (2) Items that are held for the next processing cycle.

HPTS. High Performance Transaction System. See *ImagePlus High Performance Transaction System.*

HSRR. High-speed reject re-entry.

I

image. The captured facsimile (picture) of an item represented in digital form suitable for computer processing and storage, and visual display to an operator.

ImagePlus High Performance Transaction System (HPTS). An IBM system that adds image processing capabilities to document processing.

ImagePlus HPTS Application Library Services. An IBM licensed program that supplies the HPTS system with services such as communication, data-storage management, recognition facilities, data compression, data reconstruction, and device support. The program consists of Image Host Application services, Image Processor Recognition Services, and Image Workstation Application Services.

import/export. The sending of information (export) from one system or application and the acceptance of information (import) by another system or application.

inclearings/inwork. A UK term describing checks and credits drawn on your financial institution. Similar to the term “on-us.”

incoming sequence number. A number that defines the incoming sequence of an item within the input stream. This unique number is associated with the item throughout the whole cycle of computer processing.

input string (I-string). A string of documents created by the MICR task. On each document processor run, an I-string is created. The string includes every document read by the document processor, including control documents and rejected documents. Related information, such as the pocket selected, is also stored in each record. The string also includes internally generated control records.

inscribe. Synonym for *encode.*

inscriber. A machine that encodes and inscribes in a particular format. Synonym for *encoder.*

interbank settlement sheet. A UK interbank report, produced by Inwork DCV Reconciliation, summarizing the Inwork DCV totals and the settlement figure.

Inwork. A UK term for incoming on-us work from other banks or institutions.

Inwork DCV Detail Report. A UK term for a report produced by Inwork DCV Reconciliation for each responding bank listing the DCVs and WDs that are being returned.

Inwork DCV Recapture File. A UK term for a file created by Inwork DCV Reconciliation by recapturing the Inwork DCVs and WDs after balancing. This file is matched against the Inwork DCV Summary File to produce the Inwork DCV Reconciliation File.

Inwork DCV Reconciliation File. A UK term for a file created by Inwork DCV Reconciliation by matching the Inwork DCV Recapture File against the Inwork DCV Summary File.

Inwork DCV Reconciliation Report. A UK term for a report produced by Inwork DCV Reconciliation that lists the free and missing Inwork DCVs detected.

Inwork DCV Summary File. A UK term for a file created by DKNIDCS after the completion of Prime Balancing. It contains details of all DCVs and WDs captured in the Inwork cycle and is input to Inwork DCV Reconciliation.

interface. A named and shared boundary between two functional units, (for example, component interface, subcomponent interface) defined by functional characteristics, or other characteristics, as appropriate.

invocation. Any method of starting a function within a component, subcomponent, or module, such as a direct call with parameters, use of a queue, or event control blocks (ECBs).

inwork. Checks and credits that are drawn on the financial institution that is processing them. Also termed “on-us.”

I-string. Input string.

item. A check, deposit slip, or other machine-readable document.

item-sequence number. A number that defines the sequence of an item within the input stream. This unique number is associated with the item throughout the entire cycle of computer processing.

J

jam. A condition that exists when items form a blockage anywhere in the transport mechanism of a document processor.

JGC. Joint Giro Credit.

job control language (JCL). A control language used to identify a job to an operating system and to describe the job's requirements.

JCL. Job Control Language.

JES. Job entry subsystem.

job entry subsystem (JES). A system facility for spooling, job queuing, and managing input and output.

joggler/jogger. A device that straightens and aligns items before high-speed sorting, principally to line up the lower edge and right side of a group of documents. This device is an integral component of some document processors.

Joint Giro Credit (JGC). A UK credit that may be paid in either through a clearing bank or through a post office. The two JGC types are (1) long joint giro, and (2) short joint giro. The only difference between the two types is that the long version has an Amount Due field and the short JGC does not.

K

kill. To process items to a point where no further distribution is required. See also *remit* .

kill bundle. A group of items in a kill pocket, delineated by divider slips, that forms a batch or remittance to another bank. With concurrent kill, this group can span strings. See also *remittance list.*

kill list. A document that accompanies a kill bundle, listing detail and controls for the items.

kill pass. A pass on which items are distributed to their endpoint pockets.

kill pocket. A document-processor pocket assigned to items that are sent and remitted to another bank or destination without further sorting.

L

legal tender. Any money that must, by law, be accepted in payment of debts. A personal check is not legal tender.

link-edit. To use a linkage editor to create a loadable computer program.

listed and not enclosed. A condition that exists when an item is listed on an incoming remittance/kill list or inscriber tape but is not enclosed in the kill bundle.

logical unit (LU). A port through which a user accesses SNA-network functions to communicate with another user on the network.

low-speed transit. The manual sorting and processing of checks.

LU. Logical unit.

LU 6.2. Logical unit 6.2 protocol.

LU 6.2 protocol. An SNA service that receives requests from users and from the system services control point. This service provides session management and other services for sessions between two logical units.

M

magnetic ink character recognition (MICR). The reading of magnetically encoded data on the 5/8" clear band that runs along the bottom of a document. The MICR system uses ten specially coded digits and four special symbols.

Management Information System (MIS). A DB2 system that maintains data on overall check processing. This is a subcomponent of ImagePlus HPTS Application Library Services (IALS).

manual restart. The process of physically finding and rebatching, before resuming an interrupted entry, the items to be recaptured.

mass data set (MDS). A file that contains records of all active document strings. This file consists of two direct access data sets: a directory index and a data record set.

master list. A list of all items that are read during a computer pass.

MDS. Mass data set.

merged string (M-string). The M-string, produced by DKNMRGE, represents the merging of images from the prime-pass I-string with corrected reject data. Reports that result from the M-string let you reconcile and balance input to ensure that all items were captured.

MICR. Magnetic ink character recognition.

microfilm number. The assigned item number that is also captured on microfilm.

MIS. Management Information System.

misread. A condition that occurs when a document processor interprets a character as a good character other than that which actually appears on the document codeline. Synonymous with *substitution.*

missort. An item that is found in a pocket other than the pocket to which it was sorted. This might be the result of a misread.

M-string. Merged string.

Multiple Virtual Storage (MVS). An operating system that consists of MVS/System Product (MVS/SP)* , MVS/ESA*, and the MVS Data Facility Product operating on a System/370 processor.

O

OCR. Optical character recognition.

OLMS. Online manual split.

OLRR. Online reject re-entry.

online fine sort. A computer-controlled sorting of documents (for example, checks) by either or both the account number and the serial number sequence for filing. This process commonly uses codeline data match techniques.

online manual split (OLMS). The process that sorts reject data from the MDS to produce remittance/kill lists and branch reports in the same sequence as manually sorted rejects.

online reject re-entry (OLRR). Manual entry or correction of MICR data through a display terminal.

on-us. Documents belonging to a bank that are sent to its clearing center from other banks or financial institutions. See also *inwork.*

Optical character recognition (OCR). Character recognition that uses optical means to identify graphic characters.

optional field 1. An optional, encoded field used by some US financial institutions for check truncation. It can also be used for other internal purposes.

out-clearing. A UK term meaning the sorting of documents to external destinations. The US term is *transit*. See also *outwork.*

outgoing sequence number. A sequence number or unique identification assigned to each item, identifying the kill bundle in which the item left the financial institution.

outwork. Documents that when processed leave the bank for collection from other institutions. See also *out-clearing.*

Outwork DCV Detail Report. A UK term for a report produced by Outwork DCV Reconciliation for each responding bank. It is essentially a listing of the Outwork DCV Reconciliation File.

Outwork DCV File. A UK term for a file produced by Remittance (Kill) processing. It is essentially an electronic version of the Outwork DCV Report and is used to power encode DCVs.

Outwork DCV Interbank Settlement Sheet. A UK term for a report produced by Outwork DCV Reconciliation for each responding bank, summarizing the agreed DCV totals and the figure for settlement.

Outwork DCV Recapture File. A UK term for a file created by Outwork DCV Reconciliation by recapturing the DCVs returned by other banks. This file is then

* Trademark of IBM

matched against the Outwork DCV Summary File created on the previous day.

Outwork DCV Reconciliation File. A UK term for a file created by Outwork DCV Reconciliation by matching the Outwork DCV Recapture File against the Outwork DCV Summary File.

Outwork DCV Reconciliation Report. A UK term for a report produced by Outwork DCV Reconciliation for each responding bank listing the missing and free DCVs detected.

Outwork DCV Report. A UK term for a report produced by Remittance (Kill) processing. It is similar to a CPCS-I cash letter and summarizes a number of kill bundles. It is not sent with the documents but is used to manually encode DCVs.

Outwork DCV Summary File. A UK term for a file produced by Remittance (Kill) processing. It contains a record for every Remittance (Kill) bundle processed and is grouped by endpoint within a cycle. It is used as input to Outwork DCV Reconciliation when the DCVs are returned by the responding bank on the following day.

P

pass. A single reading and sorting of a group of checks and control documents on a document processor.

pass-to-pass control. A process that maintains the total amount and item control of a group of documents on subsequent passes, when control has been established on the previous pass.

path. The path of a functional unit of work is the ordered list of tasks processed for the associated CPCS-I string. See also *flow code* and *flow control.*

pending status queue. A first-in-first-out System Manager queue through which CPCS-I applications interface to the System Manager, in sequence, to perform UOW creations, deletions, inquiries, and updates.

piggyback item. An item that was missing from its assigned pocket in a sorter and sorted “free” to an unidentified pocket, as when one document attaches itself to or overlaps another during processing.

pocket 1-1. See *system reject pocket.*

PRAD. Propagation of Adjustments.

presenting bank. A UK term for the bank sending documents and DCVs and requesting funds for the DCVs.

prime pass. The first pass of an entry on a document processor.

printing after the fact. See *deferred printing.*

process control field. Used in the US by the payor bank to know which process applies to each item. In the UK this field is called *transaction code* and is used to identify document types.

proof. Receives checks that come from tellers, mail and night depository, and internal departments of the financial institution. Proof balances transactions and inscribes or encodes the monetary amount in MICR.

proof of deposit. The act of totalling items at the deposit level and ensuring that the total of the credits equals the total of the debits.

propagation of adjustments. The process of ensuring that adjustments made in Balancing and elsewhere are carried forward to kill/remittance and other system output processes.

R

RACF. Resource Access Control Facility.

RBA. Relative block address.

reconcile. To find and correct the cause of a difference between two sets of totals.

reconciliation. See *balancing.*

reconditioning. The process of straightening folded items, inverting upside-down items, flipping reversed items, and removing any residual staples or rubber bands.

reference. A UK term for a field encoded on credit documents, corresponding to the 6-digit Serial field on debits. The Reference field may be up to 18 digits in length and (if printed in OCR) may contain alphanumeric characters.

rehandle pocket. A document processor pocket that receives items for multiple endpoints. Items directed to rehandle pockets are processed again on a later pass.

reject. A document that cannot be read in its entirety by a document processor or that fails certain editing checks. This document is normally directed to a special pocket called a reject pocket.

reject string (R-string). Strings that are created by the online reject re-entry task. Each R-string represents checks that have been re-entered online. R-strings are input to the DKNMRGE task.

relationship. Shows the parent/child hierarchy of units of work.

relative block address (RBA). In CPCS-I, the calculated location of a specific record.

remit. A UK term; to send items to another financial institution.

remittance file. A UK term for an MVS data set that is created by Remittance (Kill) processing. It is essentially an electronic version of the remittance list and may be used to support DEFT input processing at the receiving institution.

remittance list. A UK term for a CPCS-I Kill List that is produced to support negotiation and settlement of a batch of documents prefixed by a DCV. It is used for conventional interchange between clearing operations.

repass. See *rehandle pocket.*

rerun. A group of items that are sorted into a pocket on one pass and later brought into a document processor for more sorting.

Resource Access Control Facility (RACF). An MVS security subsystem that determines the validity of each operator's ID password and that controls operator access to application tasks and transactions.

responding bank. A UK term for the bank making payment on documents/DCVs received from the presenting bank.

restart. An initiation of the CPCS-I system after a system failure. A restart is generally used to start the system (after an abnormal end of a task) to cause the executive routines to re-establish the system to the status that existed before the failures.

restart buffer. An area where records are stored in an IBM 389x/XP Series document processor during online operations until they are sent to the host. The buffer is accessed during automatic restart.

resynch document. A control document used in DEFT processing to match DEFT data to the documents currently being processed on Prime and also used to separate and identify kill bundles on output.

return item. A check that is not honored by the maker's financial institution and that is returned to the depositor's financial institution.

routing/transit number field. An encoded check field that represents the financial institution on which the check is drawn. In the UK, this is referred to as the *Sort Code*.

R-string. Reject string.

S

SCI. Stacker Control Instruction.

scroll. The ability to use the DKNSCRL application to page through or look at the scroll data set. This data set includes supervisor terminal messages and DKNATASK log messages.

SDE. String directory entry.

separator. See *divider slip.*

sequence number. A number, assigned to a document, that uniquely identifies its position in a group of incoming or outgoing work.

serial field. A UK term for the 6-digit field, (equivalent to the check number in the US), which is normally the serial number of a check. On credits, the same field is called a Reference and may be up to 18 digits in length.

settlement. The act of bringing sets of related figures from two financial institutions into agreement. Adjustments are made to offset the differences.

simulated sorter. A CPCS-I facility that allows a user to run MICR, using an input file without a physical sorter.

slip. A slip is a control document used to prefix bundles for control purposes.

SMOF. System Manager Online Functions.

SNA. Systems Network Architecture.

sort code. A UK term for the field (equivalent to the routing transit field in the US) which identifies the bank and branch to which a debit or credit item belongs. It is in the format *BB-bbbb*, where *BB* identifies the bank, and *bbbb* identifies the branch within that bank. It may be printed in MICR (on checks and some credits) or in OCR (on some credits). If printed in MICR, the two parts of the field are separated by a dash (SS4).

sorter station (also document-processor station). A work station consisting of a document processor and a terminal for operator communications. Synonym for document-processor station.

sort pattern. A table used by the sort routine to determine the pocket to which a check is to be directed.

sort-pattern definition file. A collection of records that contains control information that MICR in CPCS-I uses to set up and control document sorting; it also contains data about endpoints.

sort routine. A time-dependent routine that does all processing required to direct a document to a specific document processor pocket.

sort program. A routine that performs all processing required to select a document to a pocket.

spool data set. A data set used to store printed output lines. Each spool (Simultaneous Peripheral Operations On-Line) data set is written by a CPCS-I application task and is read by the CPCS-I output writer as it is being printed.

SSB. String status block.

SSM. String segment map.

Stacker Control Instruction (SCI). SCI is the name of a language used to write programs to control the sorting of documents on a 389x document processor.

statistics. The processing of unit-of-work (UOW) data through a statistical program such as the ImagePlus Application Library Services (MIS) system. This term can also refer to the processing of unit-of-work data through a user-written statistical program.

string. The data records representing a group of items, for example, an I-string, a D-string, or an M-string. See related definitions for details.

string segment map (SSM). One of three types of segment maps in CPCS-I. Each string in the system is associated with a string segment map. Each bit in a map represents a segment of direct access storage.

string status block (SSB). This CPCS-I control block is maintained by the MDS programs for every open string.

STV. Subtotal voucher.

subcomponent. Functional subset of a component where subsetting is appropriate based on data use, logic flow, or other factors relating to modules.

subcomponent internal data. All data accessible to any modules within this particular subcomponent, but not accessible to any part of the system outside this subcomponent.

subsequent pass. A pass on which previously sorted items are resorted for further distribution.

subset. A defined portion of an entry, indicated by one or more tracer groups.

subset processing. Processing a portion of an entry beyond the document-entry step before the whole entry is run through the document processor.

subset string. A predefined group of data records that represents a portion of the physical items in an entry. A subset string can contain multiple tracer groups.

substitution. See *misread.*

subtotal voucher (STV). An optional UK document that can be inserted into a batch of documents to mark the point at which a cumulative subtotal is printed on the accompanying remittance list.

supervisor. (1) An MVS term used to refer to the system nucleus in internal storage. (2) A person responsible for operation of a financial institution area.

supervisory terminal. A special terminal or operating mode used in CPCS-I.

System Manager. A subsystem of CPCS-I that directs and controls the operations.

System Manager Online Functions (SMOF). A set of application-level tasks that monitor and modify the queues and databases of System Manager.

system reject pocket. The first physical pocket on the document processor. It is used by CPCS-I to hold machine and user-selected rejects.

System Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration of, networks.

T

tab key. A keyboard function key. The tab key causes the cursor to position to the next colon on the screen or to the top of the screen.

task. A CPCS-I application or function. User-written tasks must be in the CPCS-I BLDL list.

task list. The ordered list of tasks to be performed for a unit of work. It is determined by selecting the flow code for a given flow control record.

task status. A representation of what will happen, what is happening, or what happened during processing of this unit of work. Can be pending, ready, or complete. See related definitions for details.

total system. A system in which the computer is used for all phases of an operation.

tracer. A check-processing document used to provide pass-to-pass control.

tracer group. A grouping of documents between sets of tracers for control purposes. If subset processing is

in operation, this tracer group normally becomes a unit of work that can be processed independently of other units of work within that entry.

tracer ID. The tracer group and slip numbers corresponding to a tracer slip.

transaction code. A UK term for the 2-digit field that identifies debit, credit and control document types (similar to the Process Control Field in the US). A blank transaction code is a valid identifier for a check.

transit. The sorting of checks to external destinations. See also *out-clearing* and *outwork.*

U

unit of work (UOW). A logical entity that the System Manager uses to track a piece of work through CPCS-I. It can be informational or functional. See also *functional unit of work.*

UOW. Unit of work.

UOW status. This status represents the state of a unit of work and its associated string. Can be pending, ready, or complete.

V

Virtual Storage Access Method (VSAM). An access method for indexed or sequential processing of fixed or variable-length records on direct access storage devices.

Virtual Telecommunications Access Method (VTAM). A set of programs that control the communication between terminals and application programs.

VSAM. See *Virtual Storage Access Method.*

VTAM. See *Virtual Telecommunications Access Method.*

W

warm start. An initiation of the CPCS-I system, causing the contents of the MDS and the control data sets to be retained. A warm start is generally used for restarting CPCS-I after a normal ending.

WD (wrongly delivered). A UK term for items (debits or credits, not DCVs) that have been dispatched to the wrong bank. They are returned rather than redirected.

XREC. The dynamic control block that maps the string data at various points in the system. It cross-records or maps the string as it is in the data base, or as it is in the data space.

work. Any document or group of documents that CPCS-I processes.

work flow. An ordered list of tasks for a specific CPCS-I string. Each CPCS-I string must have a work flow.

Z

zero-balancing. The procedure that ensures that generated totals for a group of items plus any documented errors minus the control total equals zero.

Numerics

3890/XP Document Processor. A document processor in the 3890/XP Series of document processors that can read and sort documents at a rate of up to 2400 documents per minute.

3890/XP Series document processors. A series of high-speed document processors that can read and sort up to 1000, 1700, or 2400 documents per minute. These document processors include the IBM 3890/XP Document Processor, the IBM 3891/XP Document Processor, and the IBM 3892/XP Document Processor.

3891/XP Document Processor. A document processor in the 3890/XP Series of document processors that can read and sort documents at a rate of up to 1700 documents per minute.

3892/XP Document Processor. A document processor in the 3890/XP Series of document processors that can read and sort documents at a rate of up to 1000 documents per minute.

3892/XP Power Encoder Feature. An optional device that can be attached to the 3892/XP Document Processor to encode the MICR codeline field on a document.

99 M-string. See *balanced M-string.*

Bibliography

The publications in this bibliography contain information related to CPCS-I.

ACF/VTAM Publications

The following publications are related to the ACF/VTAM Version 3 Release 4 product:

IBM ACF/VTAM Programming, SC31-6436

IBM Planning and Reference for NetView, NCP, and VTAM, SC31-6124

IBM VTAM Programming for LU 6.2, SC31-6437.

Document Processor Support Publications

The following publications are related to document processor support:

IBM 3890/XP Series Document Processor General Information, GA34-2012

IBM 3890/XP Series Programming Guide, GC31-2662

IBM 3890/XP Series SPXServ Reference, GC31-2704

IBM 3890/XP MVS Support and 3890/XP VSE Support Program Reference, SC31-2654

High Performance Transaction System Publications (Version 1)

The following publications are related to the IBM ImagePlus High Performance Transaction System (Version 1):

IBM ImagePlus High Performance Transaction System General Information Manual, GC31-2706

IBM ImagePlus High Performance Transaction System Application Library Services Programming Reference, SC31-2794

IBM ImagePlus High Performance Transaction System Installation Guide, SC31-3943

High Performance Transaction System Publications (Version 2)

The following publications are related to the IBM ImagePlus High Performance Transaction System (Version 2):

IBM ImagePlus High Performance Transaction System Planning Guide, GC31-4005

IBM ImagePlus High Performance Transaction System Installation Guide, GC31-4006

IBM ImagePlus High Performance Transaction System Application Library Services Operations Guide, SC31-4010

IBM ImagePlus High Performance Transaction System Application Library Services Programming Guide, SC31-4011

IBM ImagePlus High Performance Transaction System Application Library Services Programming Reference, SC31-4012

MVS Publications (Version 5)

The following publications are related to MVS:

IBM MVS/ESA Programming: Extended Addressability, GC28-1468

IBM MVS/ESA Installation Exits, SC28-1459

IBM MVS/ESA Initialization and Tuning Reference, SC28-1452

IBM MVS/ESA JCL User's Guide, GC28-1473

IBM MVS/ESA System Messages, Volume 1 (ABA-ASA), GC28-1480

IBM MVS/ESA System Messages, Volume 2 (ASB-EWX), GC28-1481

IBM MVS/ESA System Messages, Volume 3 (GDE-IEB), GC28-1482

IBM MVS/ESA System Codes, GC28-1486

IBM MVS/DFP Version 3 Release 3: Utilities, SC26-4559

To help you find other MVS library references for various release levels, check:

MVS/ESA Library Guide for MVS/ESA System Product Version 4, GC28-1601

MVS/ESA System Product Library Guide with JES2, GC28-1423

RACF Publications

The following publications are related to RACF:

IBM Resource Access Control Facility Command Language Reference, SC28-0773

IBM Resource Access Control Facility Security Administrator's Guide, SC28-1340

IBM System Programming Library: Resource Access Control Facility, SC28-1343.

IBM Resource Access Control Facility Master Index, GC28-1035

CPCS Enhanced System Manager Publication

The following publication is related to Enhanced System Manager:

IBM Check Processing Control System: Enhanced System Manager User's Guide, SC31-4002

Index

Numerics

2-byte addressing 2-83
4-byte addressing 2-83

A

add tracer data 1-65
address space control mode testing/setting 3-26
addressing mode testing/setting 3-18
ADJ 1-129
adjustment code descriptions
  accessing, by ICDR exit 2-48
  accessing, by ODCR exit 2-99
ALDSN 3-4
ALLO 2-4
ALLOC 3-10
allocate new tracer group 1-65
allocating tracer group 1-70
AMODE 3-18, 3-37
ANTG 1-65
APCB entry
  adding to KNBLDL member 1-3
APTCB 1-60
ASC 3-26
ATD 1-65

B

bank control file
  exit routine specification
    CDIF exit 2-7
    CYCL exit 2-12
    DFTI exit 1 2-15
    DFTI exit 2 2-19
    DFTO exit 2-23
    FSCN exit 2-34
    HCDM exit 1 2-37
    HCDM exit 2 2-39
    HCDM exit 3 2-44
    HCDM exit 4 2-46
    IDCR exit 2-48
    MDIS exit 2-50
    MICR exit 2 2-60
    MICR exit 3 2-70
    MOLRI exit 2-90
    MRGE exit 2-93
  information 1-81
BCF 1-81
BLDE (DKNBLDL online editor) 1-3
BLDL
  See DKNBLDL

C

CDIF 2-7
CDM 2-69, 2-70, 2-71, 2-74
CIMS task, defining to CPCS-I 1-3
columnar reports 1-104
CPCS-I log, sending message to 1-39
CPCSNAME 3-16
CSBU 2-11, 2-83
CYC2DSCT 1-78
CYCL 2-12
cycle
  date 1-77
  date validation 2-12
  information 1-77
  status 1-77
CYCXDSCT 2-12

D

date/time functions 4-4
DCV reconciliation reporting 2-99
DEFT input 2-15, 2-19
DEFT output 2-23
DEFTOEXT 2-23
delete tracer data 1-65
delete tracer data group 1-65
deleting strings 1-60
dequeing 4-9
determining route codes 1-21
DFTI 1-123, 1-129, 2-15, 2-19, 2-20
  initialization 2-15
  input data set creation 1-123
  processing 2-19
DFTI exit 1 2-15
DFTO 2-23, 2-24
  processing 2-23
DFTP 2-29
DFTP Exit 2-29
DFTP processing 2-29
DIDSCT assembler copybook
  CDIF exit 2-8
  DFTI exit 2 2-20
  DFTO exit 2-24
  MRGE exit 2-93
  performing string I/O 1-48
  RCVY exit 2-100
DKNADCB2 1-90, 1-113
DKNADCB3 1-97
DKNAMODE 3-18
DKNAPTCB 1-129

DKNAPTCB assembler copybook
accessing cycle information 1-77
CDIF exit 2-7
creating DFTI input data sets 1-123
dynamically allocate/unallocate permanent data set 4-11
dynamically allocate/unallocate temporary data set 4-33
generating JES reports 1-97
generating spooled reports 1-90
IDCR exit 2-48
interfacing with host codeline edit routines 1-129
issuing messages 1-39
obtaining unique sequence numbers 1-86
perform QSAM input processing 4-23
perform QSAM output processing 4-28
perform VSAM I/O processing 4-37
performing security checks 1-120
performing string I/O 1-48
performing terminal I/O 1-11
RCVY exit 2-100
start parameters for manually-attached task 1-4
tracer group update utility 1-70
DKNBCFIO 1-81
DKNBLDL 1-51, 1-113
adding a task's APCB entry 1-3
DFTI exit 1 2-15
DFTI exit 2 2-19
MDIS exit 2-50
MOLRI exit 2-89
MOLRIEX parameter 2-89
USREXIT parameter 2-50
WORK parameter 2-15, 2-19
DKNCATCB COBOL copybook
accessing cycle information 1-77
CDIF exit 2-7
creating DFTI input data sets 1-123
dynamically allocate/unallocate permanent data set 4-11
dynamically allocate/unallocate temporary data set 4-33
generating JES reports 1-97
generating spooled reports 1-90
IDCR exit 2-48
issuing messages 1-39
obtaining unique sequence numbers 1-86
perform QSAM input processing 4-23
perform QSAM output processing 4-28
perform VSAM I/O processing 4-37
performing security checks 1-120
performing string I/O 1-48
performing terminal I/O 1-11
start parameters for manually-attached task 1-4
tracer group update utility 1-70
DKNCCYC2 1-78
DKNCCYCX 2-12
DKNCDIF 1-123
DKNCDP 1-123, 2-7
DKNCPARM 2-107, 2-110
DKNCRBCF 1-81
DKNCRCDP 1-123, 2-7
DKNCRDI COBOL copybook
CDIF exit 2-8
DFTI exit 2 2-20
DFTO exit 2-24
MRGE exit 2-93
performing string I/O 1-48
DKNCRFSC 2-34
DKNCRIGN 1-86
DKNCRMSG 1-39
DKNCRTG COBOL copybook 1-64, 1-70
DKNCRTGU COBOL copybook 1-70
DKNCRTHT 2-15, 2-20
DKNCRTL COBOL copybook 1-64
DKNCRTPF 2-15, 2-20
DKNCRVSM 4-38
DKNCRZB COBOL copybook
MDIS exit 2-51
DKNCRZD COBOL copybook
CDIF exit 2-8
HCDM exit 2 2-40
HCDM exit 3 2-44
MDIS exit 2-51
performing string I/O 1-48
DKNCRZE2 COBOL copybook
DKNMASS 1-48
performing string I/O 1-48
DKNCYCI 1-13, 1-77
DKNCYCLX 2-12
DKNDADJX 2-99
DKNDADJX COBOL copybook 2-48
DKNDATE 4-4
DKNDEQ 4-9
DKNDFTHT 2-15, 2-20
DKNDFTOA 2-24
DKNDFTOC 2-24
DKNDFTPF 2-15, 2-20
DKNDFTU1 2-15
DKNDFTU2 2-19
DKNDSAT 1-123, 4-11, 4-28, 4-33
DKNDYNA 4-11, 4-33
DKNENQ 4-15
DKNENQC 4-15
DKNFILL 3-21
DKNFORM 1-90
DKNFSC 2-34
DKNGETB2 1-27
DKNGETXI 1-21
DKNHCMA0
HCDM exit 1 2-37
HCDM exit 2 2-40

DKNHCMA0 *(continued)*
HCDM exit 3 2-44
HCDM exit 4 2-46
MRGE exit 2-93
DKNHCMC0
HCDM exit 1 2-37
HCDM exit 2 2-40
HCDM exit 3 2-44
HCDM exit 4 2-46
MRGE exit 2-93
DKNHCMC7 2-40
DKNIGEN 1-86
DKNIGPRM 1-86
DKNLINK2 1-84
DKNLOGU 2-100
DKNMADJX 2-53
DKNMASS 1-47, 1-48, 1-49
DKNMAYI 1-120
DKNMDISS 2-51
DKNMDIX 2-50
DKNMDIX2 2-50
DKNMDIXC 2-51
DKNMDIXP 2-51
DKNMDXR 1-113, 1-114
SNAP dump facility 1-113
DKNMIPX 2-71, 2-74
DKNMOLRI 1-129, 2-88
DKNMRGC0 2-93
DKNOLRR 2-90
DKNPARM assembler copybook
MOLRI exit 2-89
MTASK exit 1 2-95
MTASK exit 2 2-97
SECR exit 2-102
VSMGR exit 1 2-107
VSMGR exit 2 2-110
DKNPCTLI 1-64, 1-65
DKNPDSA 4-17
DKNPDSC 4-17
DKNPDSIO 1-81, 4-17
DKNQGET 4-23
DKNQGETA 4-23
DKNQGETC 4-23
DKNQPUT 1-123, 4-28
DKNQPUTA 4-28
DKNQPUTC 4-28
DKNREGS 3-23
DKNRPP1A 1-113
DKNRPP1C 1-113
DKNRPP1X 1-113
DKNRPP2A 1-113
DKNRPP2C 1-113
DKNSBT 3-24
DKNSECRX 2-102
DKNSFSC 2-34
DKNSFSCP 2-34
DKNSMSG 1-7, 1-39
DKNSMSGP 1-39
DKNSTGD 1-60
DKNTDYNA 4-33
DKNTENT 2-102
DKNTERM2 1-11, 1-12
DKNTGUT 1-70, 1-71
DKNUAM (user area manager) 1-57
DKNUEM (user exit manager) 1-133
DKNVSCOB 4-39
DKNVSENT 2-107, 2-110
DKNVSMIO 4-37, 4-38, 4-39
DKNWTO 1-45
DKNXMODE 3-26
DTD 1-65
DTDG 1-65
dynamic allocation
ALDSN macro 3-4
ALLOC macro 3-10
of permanent data set 4-11
of temporary data set 4-33

E

end-prime status 1-77
endorse date 1-77
endorse date validation 2-12
Enhanced System Manager 1-3, 1-4
enqueing 4-15
ESDS 4-37
exit 2-4
ALLO 2-4
expanded codeline record reports 1-113
expanded format sort types 2-83

F

FFLDL 3-28
field length equates/constants 3-28
finding next DKNKB record 1-84
FPACK 3-31
FSCN 2-33, 2-34
FSCN processing 2-33
FSCNUEX1 2-33
full-page buffering 1-104
FUNPK 3-34

H

HCDM
positional match sort criteria 2-46
profile parameters 2-37
reconciliation string 2-39
unmatched item processing 2-44

HCDM exit 1 2-37
HCDM exit 2 2-39
HCDM exit 3 2-44
HCDM exit 4 2-46
HCMX001 2-37
HCMX002 2-39
HCMX003 2-44
HCMX004 2-46
High Performance Transaction System
  adjustments 2-69, 2-70
  balancing 2-69
host codeline edit routine 1-129
host codeline edit routine interface 2-88
HSRR
  MRGE processing 2-93

I

IDCR 2-48
invoking user exits 1-133
issuing messages 1-39

J

JES reports 1-97

K

kill-bundle record
  finding next available 1-84
KSDS 4-37

M

M-string distribution 2-50
MDIS 2-50
MDXREC copybook 2-100
messages, issuing
  from a COBOL program 1-39
  from an Assembler program 1-39
MICR
  BEGIN 2-54
  codeline data matching 2-69
  document processing 2-59
  document processor's edit routine and table load 2-83
  document processor's edit routine and table load 2-77
  entry begin parameter validations 2-54
MICR exit 1 2-54
MICR exit 2 2-59
MICR exit 3 2-69
MICR exit 4 2-77
MICR exits 5/6 2-83
MIPIEXIT 2-70
MOLREXIT 2-88
MOLRI 2-88
MRDBDSCT assembler copybook 1-129
MRGE 2-93
MRGX000 2-93
MTASK
  additional shutdown processing 2-97
  additional startup processing 2-95
MTASK Exit 1 2-95
MTASK Exit 2 2-97
MUPA assembler copybook
  use with DKNMOLRI 1-129
  use with MOLRI exit 2-88

N

name validation 3-16
non-XF sort types 2-77

O

ODCR 2-99
OLMS 1-123
OLRR 1-129

P

P/E 2-74
pack unsigned 3-31
pass-to-pass control information 1-70
  accessing through DKNPCTLI 1-64
  tracer group update utility 1-70
PDS I/O 4-17
permanent data set
  dynamic allocation 4-11
  dynamic unallocation 4-11
power/encoding 2-69, 2-70, 2-74
processing user areas 1-57
program exit linkage 3-37
prolog 2-78, 2-84

Q

QSAM
  input processing 4-23
  output processing 4-28
QSAM data set 1-7

R

RCVY 2-100
RDXREC copybook 2-100
recovery 2-100
register equate generation 3-23
retrieving next tracer data 1-65
retrieving tracer data 1-65
retrieving tracer data pass history 1-65

return codes
CDIF exit 2-8
CYCL exit 2-12
DFTI exit 1 2-16
DFTI exit 2 2-20
DKNADCB2 1-90
DKNADCB3 1-97
DKNBCFIO 1-81
DKNCDIF 1-123
DKNCYCI 1-78
DKNDYNA 4-11
DKNIGEN 1-86
DKNMASS 1-50
DKNMAYI 1-120
DKNMDXR 1-114
DKNMOLRI 1-130
DKNPCTLI 1-65
DKNPDSIO 4-17
DKNQGET 4-23
DKNQPUT 4-28
DKNSMSG 1-40
DKNSTGD 1-60
DKNTDYNA 4-33
DKNTERM2 1-12
DKNTGUT 1-71
DKVSMIO 4-38
FSCN exit 2-34
HCDM exit 2 2-40
HCDM exit 4 2-46
IDCR exit 2-48
MOLRI exit 2-89
SECR exit 2-103
setting upon returning to CPCS-I 1-7
VSMGR exit 1 2-107
VSMGR exit 2 2-110
returning to CPCS-I 1-7
RMODE 3-18
RNTD 1-65
RSCB 2-71
RTD 1-65
RTDP 1-65

S

SBTBL 3-24
scroll log 1-60
scroll log, sending message to 1-39
searching for string types 1-47
SECR 2-102
SECRPARM 2-102
SECRWKAR 2-102
security checks 1-120
segment padding 3-21
sending messages to user executive tasks 1-27
sequence numbers 1-86
SNAPDD ddname 1-113
sort binary table search 3-24
spooled reports 1-90
standard sort types 2-77
start parameters for manually-attached tasks 1-4
STD 1-65
store tracer data 1-65
string directory 1-47
string I/O 1-47
supervisor terminal, sending messages to 1-39
System Manager
See Enhanced System Manager

T

temporary data set
dynamic allocation 4-33
dynamic unallocation 4-33
terminal I/O 1-11
TGDSCT assembler copybook 1-70
TGDSCT assembler macro/copybook 1-64
TGPROC 1-70
TGUTIL assembler copybook 1-70
time/date functions 4-4
TLDSCT assembler copybook 1-64
tracer data set
recovering with TGUT 1-70
updating with TGUT 1-70
tracer group update utility 1-70
tracer group, allocating 1-70
trademarks used in this guide xiv

U

unpack 3-34
user area manager (DKNUAM) 1-57
user area processing 1-57
user executive tasks
defining to CPCS-I 1-3
determining route codes 1-21
ending 1-37
sending messages to 1-27
writing 1-33
communication buffers 1-35
establishing addressability 1-33
searching parameter list extension 1-34
user exit manager (DKNUEM) 1-133
user exits, invoking 1-133
user security validations 2-102
user tasks, adding to CPCS-I 1-3

V

VCOMS special PCB release user exit 2-114
verify tracer group 1-65

VEXTS special terminal logon user exit 2-115
VSAM
  CPCS-I shutdown processing 2-110
  CPCS-I startup processing 2-107
  data set, defined as CPCS-I resource 2-95, 2-107, 2-110
  I/O 4-37
  TABLE 2-107, 2-110
VSCRL scroll buffer processing user exit 2-112
VSCRL scroll initialization user exit 2-113
VSMGR 2-107, 2-110
VSMVOFL 4-38
VTASK user exits
  exit 1 VSCRL scroll buffer processing 2-112
  exit 2 VSCRL scroll initialization 2-113
  exit 3/4 VCOMS special PCB release processing 2-114
  exit 5 VEXTS special terminal logon processing 2-115
VTG 1-65

W

Write-to-Operator messages
  from a COBOL program 1-39, 1-45
  from an Assembler program 1-39

X

XF sort types 2-83
XF1MAP 2-71
XRETURN 3-37

Z

ZBDSCT assembler copybook
  MDIS exit 2-51
ZDDSCT assembler copybook
  CDIF exit 2-8
  HCDM exit 2 2-40
  HCDM exit 3 2-44
  MDIS exit 2-51
  performing string I/O 1-48
ZE2DSCT assembler copybook
  DKNMASS 1-48
  performing string I/O 1-48

Communicating Your Comments to IBM

Check Processing Control System
International MVS/ESA™
Programming Reference
Release 1

Publication No. SC31-3997-03

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States & Canada: 1-800-955-5259
- If you prefer to send comments electronically, use this network ID:

  IBM Mail Exchange: USIB1362 at IBMMAIL

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

Check Processing Control System
International MVS/ESA™
Programming Reference
Release 1

Publication No. SC31-3997-03

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC31-3997-03

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Payment Solutions
Department 58G MG96/204
8501 IBM Drive
Charlotte NC 28262-8563

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5799-FKT

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-3997-03