

Transaction Processing Facility



System Installation Support Reference

Version 4 Release 1

Transaction Processing Facility



System Installation Support Reference

Version 4 Release 1

Note!

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

Sixteenth Edition (June 2002)

This is a major revision of, and obsoletes, SH31-0149-14 and all associated technical newsletters.

This edition applies to Version 4 Release 1 Modification Level 0 of IBM Transaction Processing Facility, program number 5748-T14, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1994, 2002. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	xi
Tables	xiii
Notices	xv
Trademarks	xv
About This Book	xvii
Who Should Read This Book	xvii
Conventions Used in the TPF Library	xvii
Related Information	xviii
IBM Transaction Processing Facility (TPF) 4.1 Books	xviii
IBM Systems Application Architecture (SAA) Books	xviii
IBM High-Level Language Books	xix
Miscellaneous IBM Books	xix
Online Information	xix
How to Send Your Comments	xix
Control Program User Exits Overview	1
Exit Points	2
Changing Dynamic Exit Points to Nondynamic	3
Changing Nondynamic Exit Points to Dynamic	3
Associating an Exit Point with a Function	3
CCUEXT CSECT	3
User Exits Control List (UCL)	3
Service Routine for UXCMC Macro	4
General Post-Interrupt Routine (UXGPIR)	4
User Exit Routines (CUSR)	4
User Static Override Bitmap Table	4
Installing Control Program User Exits	4
Installing Multiple Functions in a User Exit	5
Creating a Control Program Exit Point and User Exit	5
Conditions and Considerations When Using User Exits	6
User Exit Routines - Common Entry Conditions	6
User Exit Routines - Common Return Conditions	7
User Exit Routines - Common Programming Considerations	7
Macro Servicing	8
TPF Macro Processing: Limitations/Restrictions	8
Control Program User Exits	9
Application Timeout Processing	10
BACKC Macro Entry	11
C Debugger Initialization	12
C Debugger Return	14
C Library Function Call (TARGET(TPF))	15
C Library Function Return (TARGET(TPF))	16
C Stack Exception (TARGET(TPF))	17
C Stack Exception Return (TARGET(TPF))	19
Catastrophic Recovery	20
CCCTIN (CT25 and CT26)	22
CCCTIN (CT99)	24
Control Transfer	25
Core Resident Enter/Back Macro	27

CPU External Interrupt	29
CPU Timer Interrupt	30
Create Macro Control Point	31
Create Macro Postinterrupt	32
Critical Record Filing	33
Debugger Trace Selection	34
Debugger Trace Table Entry Activation	35
DLAYC Macro Entry	36
Dump Override Table	38
Dynamic Load Module Environment Initialization (ISO-C)	39
Dynamic Load Module External Function Call	41
Dynamic Load Module External Function Call Entry	42
Dynamic Load Module Return Processing	43
Dynamic Load Module Return Processing Entry	45
ECB Creation	46
ENTxC Macro Entry	48
EXITC	49
FARF Address Generation	50
Fast Link Macro	51
Fast Link Macro Decoder	52
File-Resident Enter/Back Macro	53
General Postinterrupt Processing	55
Get Block With ECB	56
Get Block without ECB	57
Get Common Block	58
Get System Work Block	59
Library Function Call (ISO-C)	60
Library Function Return (ISO-C)	61
LODIC Macro	62
Online Mini Dump	64
Pool Address Retrieval	65
Program Event Recording (PER)	66
Program Event Recording (PER) Debugging Tools	67
RCS I/O Queue Thresholding	69
Release Block With ECB	71
Release Block without ECB	72
Release Common Block	73
Release System Work Block	74
RIAT	75
ROUTC	77
SNAPC Error	79
SNAPC Error Entry	80
Split Access	81
Split Chain Header Access	82
Stack Overflow Processing (ISO-C)	83
Stack Overflow Processing Entry (ISO-C)	84
Suspend ECB	85
Suspend List Post-Interrupt	86
Suspend List Resource Checking	87
SVC Macro (Immediate)	88
SVC Macro (Wait or Implied Wait: Postinterrupt)	89
SVC Macro Decoder	90
System Error	91
System Error Entry	92
TMSLC Macro	93
TPFAR	95

TPPDF Macro Trace Call	96
TPPDF Macro Trace Return	97
Trace C User Data	98
Trace Environment Customization	100
Transaction Log Write	101
Update Tape Display	102
User Header Label	103
User Trace Area Initialization	105
Validate Tape for Output	106
WAITC Macro Entry	107
WTOPC Message Translation	108
ECB-Controlled Program User Exits Overview	109
Exit Points	109
Selective Activate Exits	109
Activating the Selective Activation Function	109
Creating an Enable Command.	109
Creating a Disable Command	110
Data Macros to Develop User Exits	111
E-type User Exit Considerations	111
User Exit Allocation and Activation	111
ECB-Controlled Program User Exits	117
Check OLDR Load Deck	118
Clock Global Update Exits	119
Command Manager.	120
Common Symbol Table	121
Communications Source Common	122
Continuous Data Collection Information Storage	123
Continuous Data Collection Table Creation	124
CP-CP Session Activation	125
Database Reorganization	126
Deactivate Phase I Selective Activate	128
Deactivate Phase II Selective Activate	129
Deadlock Detection.	130
DEARRANGE_CTK9 (UPX1)	131
Debug Registration	133
Detect Selective Activate Support	134
Display	135
DNS Select an IP Address	136
Dump Data	137
Dynamic LU	139
ECB Display	142
Extra Program Record Report	143
FILE_CY2KT (UPX7)	144
FILE_STCCR (UPX3)	145
FIND_CY2KT (UPX6)	147
FIND_STCCR (UPX2).	148
Get Global Environment Lists	149
Loadset History	150
Log Recovery Error Processing	151
LU Registration	152
LU 6.2	153
MATIP ASCU List	154
MATIP Assign LNIATA.	155
MATIP Flow ID	156

MATIP Host Name	157
MATIP Router	158
MATIP Security	159
MATIP Session Start	161
MATIP Translation	163
Message Queue Interface (MQI) Channel Exits	164
Message Router	165
Module Copy Selection/Validation	166
Nonsocket Activation	167
Nonsocket Connect.	168
Nonsocket Deactivation	169
Nonsocket Message	170
Output Message Filtering.	171
Output Message Re-formatting	172
Program Event Recording (PER).	173
Program History	174
REARRANGE_CTK9 (UPX0)	175
Recoup Command	177
Recoup Phase 1	178
Recoup Restart	179
Segment URS1	180
Segment URS2	181
Segment USC1	182
Segment USC2	183
Segment USC3	184
Segment USC4	185
Select A Host	186
Select ALS to Adjacent APPN Node.	188
Select an RTP Connection	189
Select TCP/IP Support	191
Selective Activate Message Router	192
Selective Activate Restart	194
Selective Activate Structure Initialization	195
Selective Activate Structure Update	196
Selective Core Resident Load	197
Selective Recoup	198
SLC Communication Source	199
SNA Communication Route Selection	200
SNA Message Recovery	201
SNMP Enterprise-Specific MIB Retrieval	202
SNMP Manager Validation	203
Socket Accept.	204
Socket Activation.	205
Socket Connect	206
Socket Cycle-Up	207
Socket Deactivation.	208
Socket System Error	209
System Error Message	210
Tape Display Setup	211
Tape Library Validation	213
TCP/IP Native Stack Support Accept Connection	214
TPF File System Initialization	215
TPF MQSeries Assign LNIATA.	216
TPF MQSeries Channel Message	217
TPF MQSeries Channel Message Retry	219
TPF MQSeries Channel Security	221

TPF MQSeries Convert to Object Handle	223
TPF MQSeries Queue Trigger	224
TPF MQSeries Start Queue Manager	225
Trace-by-Terminal	226
User Command Processor	227
User Data Recovery Copy Support	228
User Data Recovery Restore Support	229
User Device	230
User Global Symbol Table	231
User Label Routines	233
User Library Function	235
User Symbol Override Table	236
VFA Restart	238
Virtual IP Address Processor Deactivation	239
Virtual Reader.	240
WTOPC Page Control.	241
3270 Welcome Screen	242
Global Area	243
Terminology	243
Structure of the Global Areas	244
Global Area 1 (GL1)	245
Global Area 2 (GL2)	247
Global Area 3 (GL3)	247
Global Area 4	247
Components of the Global Area	247
Global Records	247
Global Directories	248
Global Blocks	249
Global Fields	250
System Environment Considerations	250
Multiple Database Function Environment	250
Tightly Coupled Environment	252
Using the Global Area by Applications	255
GLOBZ: Define Global Fields Macro	255
GLMOD: Change Global Protect Key Macro.	256
FILKW: File Keyword Macro	257
GL0BA: Define Global 1 Macro	257
GL0BY: Define Global 3 Macro	257
SYNCC: The Global Synchronization Macro.	258
Programming Considerations	258
Defining Addressability to Globals	258
Global Area Requirements of the Control Program	260
Loading Globals	261
SIP for Globals	261
Creating the Input Data Set.	261
Considerations for Preparing Input	305
Synchronization of Globals	306
Requirements for Synchronization	307
Locating Global Areas in a Dump.	307
Examples of I-Stream Shared and Unique Globals	309
Main Storage Super GOA Copy	312
Main Storage Prime GOA Copy	313
Sample STC Card Images for Global Block Creation	314
Examples of Coding the SYNCC Macro	317

Loaders	321
General File System Components	321
Changing CIMR Components	322
Loading System Components to a New TPF System	322
Initializing the General File	322
Formatting the General File and Online Modules	322
Creating a General File Load Deck	322
Loading System Components to the General File	338
IPLing the General File	339
Loading Fixed-File Records	340
Loading System Components to an Existing TPF System	341
Creating a New Fallback Image	341
Creating an Auxiliary Load Deck	342
Loading System Components to a Storage Medium	358
Loading from a Storage Medium to the TPF System	360
Enabling an Image	361
Moving Keypoints to the Working Area	361
IPLing an Image	362
Loading E-Type Programs to an Enabled System	362
Creating an E-Type Loader Load Deck	362
Loading E-Type Programs to a Storage Medium	367
Loading and Activating a Loadset of Programs	368
Loading a Loadset of Programs	368
Allocate Programs That are Unallocated	369
Activating a Loadset to Test New Programs	369
Using Loadsets	369
Accepting a Loadset of E-Type Programs	372
Using E-Type Loader Functions	372
 Record ID Attribute Table	 373
Contents of the RIAT	373
Addressing the RIAT	374
Programming Areas	374
Programming Techniques	374
Record Size	374
Frequency of Access	374
Record Life	374
Record Generation	374
References	375
 Multiple Assembly/Compilation Print Program	 377
Printing Multiple Assembly and Compilation Listings	377
JCL Control Cards	377
Return Codes	378
Error Messages	378
Hardware Requirements	379
 Macro Cross-Reference	 381
Specifying the DCRS Search Parameters	381
Example of DCRS	382
Specifying a DREF Heading Parameter	382
Input to the Macro Cross-Reference Programs	382
DCRS Program	383
SORT Program	383
DREF Program	383
Control Cards	383

Procedure	384
Output from the Macro Cross-Reference Programs	384
Listings	384
Files	384
DCRS Attention Messages	384
DCRS Error Messages	385
DREF Messages.	386
References	386
Multiple Assembly/Compilation Program	387
Input	387
Files	387
JCL Control Cards	387
User Considerations	392
Output	393
Listings	393
Files	393
Sample JCL	393
Printout Directory to Listings	393
Assembly Listings to Tape	394
E-Type Assemblies to Tape	395
E-Type Compilations Sent to the Printer	396
E-Type Compilations Sent to the Printer	397
Error Messages	398
Hardware Requirements	398
System Allocator	399
Allocating Programs, Transfer Vectors, and Pools.	400
Creating the Input Deck	401
Comments	401
Specifying the Addressing Mode	401
Allocating Programs	402
Allocating Transfer Vectors	404
Allocating Spare Program Slots	405
Defining Pools.	405
Adding Your Own Function Switches	406
Running the System Allocator (SALO)	406
To Run Only SIP Stage I.	406
To Run Both SIP Stages I and II	407
Creating the Program and System Allocator Tables	407
Variable Cross-Reference Listing	409
JCL Control Cards	409
Control Card Input	411
Scan the Entire PDS for Every Type of Variable	412
Scan a List of Members	412
Subset of Variable Types Option	412
Print the Globals Found in the Order Specified.	413
Option Defaults	413
Examples of Control Card Options	413
Procedure	414
Output	414
Error Messages	414
Hardware Requirements	415
References	415

Appendix. JCL Load Deck Examples	417
TLDR JCL to Load Components to GDS	417
TLDR JCL to Load Components to Tape	417
TLDR JCL to Load Components to VRDR	418
OLDR JCL to Load E-Type Programs to GDS	419
OLDR JCL to Load E-Type Programs to Tape	419
OLDR JCL to Load E-Type Programs to VRDR	420
Index	421

Figures

1.	Relationship of TPF Function, Exit Point, and User Processing	2
2.	Selective Activation Table Example	110
3.	Selective Activation Index Example.	110
4.	Global Storage Allocation for a Base-Only System with a Single I-Stream	246
5.	Global Storage Allocation for a Single I-Stream with 2 Subsystems and 5 SSUs	251
6.	Global Storage Allocation for 3 I-Streams with 1 Subsystem (the BSS) and 2 SSUs.	254
7.	GOA Data Structure	262
8.	Super GOA Layout	302
9.	STC Input Using Load Modules	303
10.	SSUT Entry	309
11.	Example of the Super GOA	313
12.	Example of a Prime GOA	314
13.	ALDR Run Load Deck Example	323
14.	ALDR Input Control Load Deck Example	325
15.	ALDR Input Control Load Deck Example Showing FCTB in Program Object Format from HFS	326
16.	General File Load Using the General File Loader Offline Segment (ALDR)	338
17.	General File IPL Using the General File Loader Online Segment (ACPL).	339
18.	Loading Fixed-File Records Using the ZSLDR Command	340
19.	TLDR Run Load Deck (to Tape) Example	343
20.	TLDR Input Control Load Deck Example	346
21.	TLDR Input Control Load Deck Example Showing FCTB in Program Object Format from HFS	347
22.	Auxiliary Load via Auxiliary Loader (Offline — MVS)	358
23.	Sample Summary Listing for Auxiliary Loader (with Key to Output Listing)	359
24.	Auxiliary Load via Auxiliary Loader (Online — TPF).	360
25.	OLDR Run Load Deck (To GDS) Example	363
26.	E-Type Load via E-Type Loader (Offline — MVS)	367
27.	E-Type Load via E-Type Loader (Online — TPF)	368
28.	Printout Directory to Listings JCL	394
29.	Assembly Listings to Tape JCL	394
30.	E-Type Assemblies to Tape JCL	395
31.	E-Type Compilations Sent to the Printer JCL	396
32.	E-Type Compilations Sent to the Printer JCL	397
33.	Operation of SALO	400
34.	Sample JCL Created by Running SIP Stage 1	407
35.	JCL Required to Run VCRS	410

Tables

1.	Activation and Allocation of User Exits for ECB-Controlled Programs	111
2.	LU Types and User-Specified OSTG Options	139
3.	X.25 LU Types and the Permitted OSTG Options	140
4.	Description of Fields in the Dynamic LU User Exit	140
5.	The Byte Arrangement for Message Display	211
6.	Initial File System Values	215
7.	Summary of CLASS Attribute for File Resident Programs	403

Notices

References in this book 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 in this book 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 may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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 830A
Mail Drop P131
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any pointers in this book to non-IBM Web sites are provided for convenience only and do not in any way serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this book or accessed through an IBM Web site that is mentioned in this book.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle
Advanced Peer-to-Peer Networking
APPN
C/370
CICS
DFSORT
IBM
MQSeries
MVS/ESA
OS/390
PR/SM

VisualAge
VM/ESA.

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

About This Book

TPF system programmers should use this book and *TPF System Generation* to install the TPF system and run offline support packages. This book contains information about the following subjects:

- User Exits
- Global Areas
- Loaders
- Record ID Attribute Table
- Multiple Assembly/Compilation Print Program
- Macro Cross Reference Program
- Multiple Assembly/Compilation Program
- System Allocator
- Variable Cross-Reference Listing

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, Systems Network Architecture (SNA). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in the *TPF Library Guide*.

Who Should Read This Book

This book should be read by TPF system programmers who install the TPF system and run offline support packages.

Conventions Used in the TPF Library

The TPF library uses the following conventions:

Conventions	Examples of Usage
<i>italic</i>	Used for important words and phrases. For example: A <i>database</i> is a collection of data. Used to represent variable information. For example: Enter ZFRST STATUS MODULE <i>mod</i> , where <i>mod</i> is the module for which you want status.
bold	Used to represent text that you type. For example: Enter ZNALS HELP to obtain help information for the ZNALS command. Used to represent variable information in C language. For example: level
monospaced	Used for messages and information that displays on a screen. For example: PROCESSING COMPLETED Used for C language functions. For example: maskc Used for examples. For example: maskc(MASKC_ENABLE, MASKC_IO);
<i>bold italic</i>	Used for emphasis. For example: You <i>must</i> type this command exactly as shown.

Conventions	Examples of Usage
<u>Bold underscore</u>	Used to indicate the default in a list of options. For example: Keyword=OPTION1 <u>DEFAULT</u>
Vertical bar	Used to separate options in a list. (Also referred to as the OR symbol.) For example: Keyword=Option1 Option2 Note: Sometimes the vertical bar is used as a <i>pipe</i> (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=option
Scale	Used to indicate the column location of input. The scale begins at column position 1. The plus sign (+) represents increments of 5 and the numerals represent increments of 10 on the scale. The first plus sign (+) represents column position 5; numeral 1 shows column position 10; numeral 2 shows column position 20 and so on. The following example shows the required text and column position for the image clear card. ...+....1....+....2....+....3....+....4....+....5....+....6....+....7... LOADER IMAGE CLEAR Notes: 1. The word LOADER must begin in column 1. 2. The word IMAGE must begin in column 10. 3. The word CLEAR must begin in column 16.

Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF ACF/SNA Network Generation*, SH31-0131
- *TPF Database Reference*, SH31-0143
- *TPF Library Guide*, GH31-0146
- *TPF General Macros*, SH31-0152
- *TPF General Information*, GH31-0147
- *TPF Non-SNA Data Communications Reference*, SH31-0161
- *TPF Main Supervisor Reference*, SH31-0159
- *TPF Migration Guide: Program Update Tapes*, GH31-0187
- *TPF Operations*, SH31-0162
- *TPF Program Development Support Reference*, SH31-0164
- *TPF ACF/SNA Data Communications Reference*, SH31-0168
- *TPF System Generation*, SH31-0171
- *TPF System Macros*, SH31-0151
- *TPF Transmission Control Protocol/Internet Protocol*, SH31-0120.

IBM Systems Application Architecture (SAA) Books

- *SAA AD/Cycle C/370 User's Guide*, SC09-1763.

IBM High-Level Language Books

- *High Level Assembler /MVS & VM & VSE Programmer's Guide*, SC26-4941
- *OS/390 C/C++ User's Guide*, SC09-2361.

Miscellaneous IBM Books

- *DFSORT Application Programming Guide*, SC33-4035
- *MQSeries Distributed Queue Management Guide*, SC33-1139.

Online Information

- *Messages (Online)*
- *Messages (System Error and Offline).*

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

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 prefer to send your comments electronically, do either of the following:
 - Go to <http://www.ibm.com/tpf/pubs/tpfpubs.htm>.
There you will find a link to a feedback page where you can enter and submit comments.
 - Send your comments by e-mail to tpfid@us.ibm.com

- If you prefer to send your comments by mail, address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

- If you prefer to send your comments by FAX, use this number:
 - United States and Canada: 1 + 845 + 432 + 9788
 - Other countries: (international code) + 845 + 432 +9788

Control Program User Exits Overview

User exits allow you to add user-unique processing at various points in TPF programs without having to modify the released programs. Control program user exits are called by control program exit points that reside in control program CSECTs.

Note: User exits can affect system performance. Therefore, have detailed knowledge of TPF system internals before activating or writing user exit code.

Figure 1 on page 2 shows an overview of control program user exit processing. When the TPF control program encounters a exit point that is not active, control program processing continues. When the TPF control program encounters an active exit point, control is passed to the User Exit Interface (UXITC). The UXITC macro generates linkage to insure that the interfaces are standard among all exit points (as exit points are scattered throughout the control program). Control is then passed to the control program CSECT CCUEXT. CSECT CCUEXT, that contains the copy member CUSR, then passes control to the user exit for user processing. If the user exit routine does not cause the current ECB to give up control of the CPU, the routine can return to the NSI in the exit point interface. If the user exit routine gives up control of the CPU, user processing is suspended for the ECB and the stack pointer (R13) and the contents of the stack area are lost.

Note: The user exit must restore the stack pointer (R13) and registers before it returns to the NSI in the exit point.

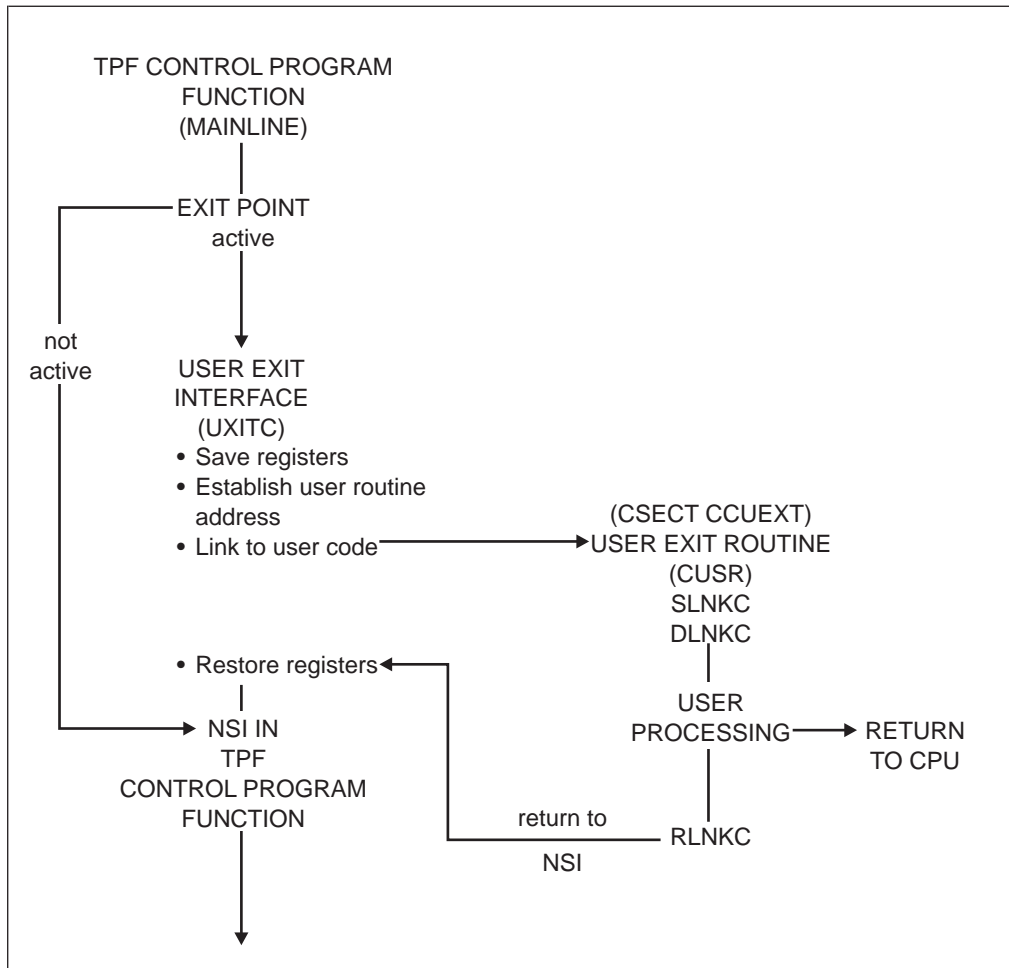


Figure 1. Relationship of TPF Function, Exit Point, and User Processing

If the user wants to resume processing the ECB later, the ECB can be placed on a CPU loop (task dispatcher) list with the Postinterrupt address label, UXPI. This causes control to be passed to the General Post Interrupt routine (UXGPIR in CSECT CCUEXT) which reactivates the ECB when it is removed from the task dispatcher list (see “General Postinterrupt Processing” on page 55).

Note: A direct return to the CPU Loop from the System Error user exit MUST NOT be attempted. An interface has been provided for this purpose. For more information, see “Programming Considerations” comments for the System Error user exit in the CSECT CCUEXT code.

Exit Points

Exit points are predefined locations in TPF system processing from which user-unique processing code can be invoked. This user-unique code (called *user exit*, *user exit routine*, and *user processing*) will then execute as an extension of a TPF system function.

Exit point status is not carried across an IPL. The ZSTIM command can be used during restart, cycle-up, or cycle-down to automatically activate or deactivate exit points.

A dynamic control program exit point is one that can be activated or deactivated by an online macro (UXCMC) issued from a program. A nondynamic control program exit point does not change. It is active or inactive. You cannot change it without reinitializing the system.

Changing Dynamic Exit Points to Nondynamic

Note: Dynamic overlay exits (exits that change the control program code with exception code) cannot be changed to nondynamic.

To change a dynamic exit point to a nondynamic exit point, you must do the following:

1. Change the corresponding entry in the &ATT (attribute) table of the DCTUCL macro to "UCLXPA" (nondynamic and active).
2. Assemble CCUEXT and CCNUCL, and link-edit the control program.
3. Make sure that you have removed all UXCMC macro calls treating this user exit as dynamic.

Changing Nondynamic Exit Points to Dynamic

Changing nondynamic exit points to dynamic exit points requires modification of control program code (at the exit point) and any related initialization code. Therefore, it is not recommended.

Associating an Exit Point with a Function

You may want to associate specific exit points with a function (for example, test tools, accounting, data collection, and others) so the function can be activated or deactivated using a command. If you do, make sure that the exit point can be deactivated and is not being used by another function.

Multiple Functions in User Exits

You may need to implement more than one function in a user exit. If you do, macros are provided that allow you to define and control multiple functions residing in a single dynamic or nondynamic user exit.

CCUEXT CSECT

The CCUEXT CSECT contains user-written exit routines and associated TPF control mechanisms. Therefore, you do not have to update the TPF control program in many locations to apply user-written code.

CCUEXT contains the following major functional components:

- User Exits Control List
- Service Routine for UXCMC Macro
- General Post-Interrupt Routine (UXGPIR)
- User Exit Routines (CUSR copy member)
- User static override bitmap table (CUDP copy member)

User Exits Control List (UCL)

The UCL and its DSECT are generated by the DCTUCL macro. The UCL is allocated and initialized in CCUEXT. The UCL contains the following information for each exit point:

- Name of the exit point (1–4 characters).
- Index number of the exit point.
- Status of the exit point (active or not active).

- Current address of the user exit routine for this exit point.
- The default address of the user exit routine for this exit point.
- The address of the overlay parameter list (for dynamic overlay exits).
- A list of bits that define the functions that are associated with each user exit.

The current address is the same as the default address unless changed by the UXCMC macro. The default address is defined by address constants using specific user exit routine labels in copy member CUSR. The UCL is addressable in the control program (by referencing the label UCLTAB), from the control program itself, or from an E-type program (by CINFC for label CMMUCL).

Service Routine for UXCMC Macro

The entry point of the UXCMC service routine is CPMUXT. The input to UXCMC is the UX1PL DSECT parameter list. UX1PL is formatted by the caller. UX1PL designates the type of action to be taken (activate or deactivate), the exit points to be acted on, and functions in the exit to be acted on. For activate, UX1PL can designate functions to be activated and the address of the alternate user processing routine for exit points.

Note: For user exits with multiple functions, the alternate address will be used to point to an alternate stub routine. Alternate addresses will not be supported for each function.

For deactivate, UX1PL can designate functions to be deactivated.

If UX1PL is formatted correctly, the requested action is taken. If UX1PL is not formatted correctly, you receive an appropriate return code and **no** action is taken.

General Post-Interrupt Routine (UXGPIR)

The General Post-Interrupt Routine (UXGPIR) allows you to restart user-suspended ECBs. The ADDLC or ADDFC macro, with a post-interrupt address of UXPI (a routine in CCNUCL), places an ECB on a dispatch list. When the ECB is removed from the list, by the System Task Dispatcher (CPU Loop), UXPI passes control to UXGPIR. If the post-interrupt exit point is active, UXGPIR passes control to the user's post-interrupt exit routine. If the post-interrupt exit point is not active, the suspended ECB is reactivated by executing a load PSW from the PSW in the ECB field CE1PSW.

User Exit Routines (CUSR)

User exit routines are user developed code to be processed at each active exit point.

User Static Override Bitmap Table

User IDOTB macro calls associate one or more areas of main storage with a given system error. Use of the static override bitmap table and its effect on dump content are described in the *TPF Program Development Support Reference*.

Installing Control Program User Exits

To install control program user exits, do the following:

1. Update or replace copy member CUSR in CCUEXT. The IBM supplied copy of CUSR has user exit routine labels that are equated to a value of zero (EQU 0). When you install a user exit routine, remove the predefined entry point label and place it at the beginning of the user exit routine. Then add your code to the CUSR copy member.
2. Assemble and link-edit the CCUEXT CSECT. The user exit routine is then incorporated in the control program. If the user exit routine is installed for a nondynamic exit point, no more action is required and the exit routine will be invoked by the control program whenever the exit point is met.
However, if a dynamic exit routine was installed, you must also provide a program that will issue UXCMC macros to activate or deactivate the appropriate exit point in order for the exit routine to be useable.

Note: To properly code a user exit routine, the programmer must be thoroughly familiar with the internals of the control program.

Installing Multiple Functions in a User Exit

To install multiple functions in a user exit routine, do the steps shown in “Installing Control Program User Exits” on page 4 and the following:

1. Define a bit in IUXEQ for each function you want to use in a user exit. The IBM-supplied copy of IUXEQ has bits defined for vendor products that are installed in user exits.
2. Update the user exit code in CUSR with a stub routine that will link to each function defined for use by the exit point. The stub routine can be built by coding a sequence of UXMAC macros.

Note: The order of UXMAC macros dictates the order in which functions in an exit will get control.

3. Provide a program to activate or deactivate functions in the user exit using the UXCMC macro. Define these functions in the expanded format of the parameter list, UX1PL.

Note: Use this activation or deactivation process for dynamic and nondynamic user exits.

Creating a Control Program Exit Point and User Exit

The TPF system provides exit points for all of the control program user exits listed in this chapter. To create your own control program exit point and user exit, do the following:

1. Update these 3 arrays in DCTUCL:
 - a. The Exit Point List (&UEP). This array contains abbreviated names for the user exits.
 - b. The Exit Point Status array (&ATT). This array identifies whether an exit is dynamic (and inactive or active), or nondynamic. Dynamic exit points have an initial value of “UCLDYN” (dynamic and inactive), while nondynamic exit points are identified by 2 zeros.
 - c. The Exit Point Description array (&DESC) contains a description of the exit.

Entries in each array correspond to entries with the same subscript in the other 2 arrays. For example, the entries for the nondynamic RIAT exit point are:

```
&UEP(37)  SETC  'RIT'
&ATT(37)  SETC  '00'
&DESC(37) SETC  'RIAT EXIT POINT'
```

The entries for the dynamic overlay fast link macro exit point are:

```
&UEP(13)  SETC  'FLX'
&ATT(13)  SETC  'UCLDYN+UCLOVR'
&DESC(13) SETC  'FAST LINK MACRO EXIT'
```

Entries can be added by overlaying spare or unused entries or by adding to the end of the arrays. Twelve spare entries have been provided for you.

When adding entries to these arrays it is important to insert new entries at the end of the array, but before the last entry. The last entry in each array marks the end, and must have the following values:

```
&UEP(n)    SETC  'XXX'
&ATT(n)    SETC  '00'
&DESC(n)   SETC  'END OF TABLE'
```

where *n* is the size of the array, increased by the number of exits you have added.

Note: The arrays are logically split into two sections. The first section is fixed at 160 entries (index numbers 0 — 159); it is used for performance-critical user exits. Each user exit has a corresponding entry both at CPMUXTBL in the CAPT low core table and in the DCTUCL. The second section is used for all other user exits; each of these user exits has a corresponding entry only in the DCTUCL.

2. If you add a new control program user exit by overlaying a spare or unused entry, you need to reassemble only the CAPT copy member. If you add a new control program user exit by adding an entry to the end of an array, you need to reassemble **all** CP CSECTS and link-edit the control program because of CAPT copy member expansion.
3. Update one array in UXITC: the Exit Point List (&UEP). This entry corresponds to the &UEP entry in the DCTUCL macro, and the subscripts must match. Additions to this array are made in exactly the same way as for the DCTUCL arrays: the subscript is the same for the corresponding entry, and any additions must be made at the end, before the entry that marks the end of the array:

```
&UEP(n)    SETC  'XXX'
```
4. Add UXITC macro processing at the appropriate exit point, to use the new user exit.
5. Install the user exit (see “Installing Control Program User Exits” on page 4).

Conditions and Considerations When Using User Exits

This section lists the conditions and considerations when using user exits.

User Exit Routines - Common Entry Conditions

The following entry conditions apply to all user exit routines unless specified in the individual exit point interface descriptions.

1. Register 15 will contain the base address of the user exit routine.
2. Register 13 will contain the address of a stack area which can be used by the user processing to save registers or other data.

3. To preserve the integrity of the stack, user processing must **not** issue an SVC macro, since this would cause the stack register to be re-initialized and the current saved data to be lost.
4. The stack at the address pointed to by register 13 will contain the contents of linkage registers 14 and 15 when the test for the user exit is performed.
5. Register 14 is the linkage register and contains the NSI return address from the user exit routine to the exit point in the control program.
6. Registers 11 and 12 contain the fixed base register values.
7. The contents of registers 0 through 10 are not modified by the user exit interface. Their contents and meaning are not predictable and are dependent on the routine that caused the exit point to be executed.

User Exit Routines - Common Return Conditions

The following return conditions apply to all exit routines unless specified otherwise in the individual exit point interface descriptions.

1. All registers (for example, general purpose, floating point, control point, and others) which are modified by user exit processing must be restored before returning to NSI in the user exit interface.
2. The system state, mask, protection key, and address space must be set exactly the same as on entry to the user exit routine.
3. If the user exit processing does **not** return to NSI in the user exit interface (for example, the ECB is suspended), it is your responsibility to cause control to be passed to the CPU loop (Task Dispatcher) with the following conditions:
 - The fixed base registers must be set to the proper value.
 - The system must be in supervisor state.
 - The protect key must be 0.

User Exit Routines - Common Programming Considerations

1. Follow proper linkage conventions. You are not required to use the linkage macros when coding the user exit routines, but the linkage conventions **must** be adhered to. The following shows an example of linkage macros in a user exit routine:

```
* User Exit routine
UCCXXX DS 0H
        SLNKC LOREG=0,DSECT=SA01
        DLNKC
        .
        . (User Processing)
        . Can contain CLNKC calls to other user
        . routines/subroutines
        .
        RLNKC LOREG=0 returns to exit point
```

2. Registers 13, 14, and 15 should not be used by user exit processing.
3. If required, user exit processing may increase the system mask only (for example, reduce the type of interrupts allowed).
4. When a user exit is activated from the ISO-C environment, the following conditions apply:
 - The data save area (DSA) pointer (R13) is saved in an ECB.
 - R13 is set to the CP stack address.

- The contents of R8–R12 (at the time of the user exit) are saved on the CP stack.
- R8, R11 and R12 are set to common interface conditions.
- The CP stack is pushed to point to the second stack frame.

Macro Servicing

There are many points in the control program where macro servicing can begin or end. Macro servicing can begin in the following:

- SVC macro decoder
- Fast link macro decoder
- Processing an ENTxC macro
- Processing a BACKC macro
- WAITC macro processing.

Macro servicing can end in one of the following ways:

- An SVC macro gives up control and completes its processing using a postinterrupt routine (wait or implied wait).
- An SVC macro processes to completion without giving up control (for example, no wait or implied wait).
- A fast link macro return (no loss of control).

All of these points in macro servicing have unique user exit points because the system condition at each place is unique.

TPF Macro Processing: Limitations/Restrictions

The user exit points associated with macro processing do not recognize whether a macro has been issued from an E-type program or from the control program. This means that any macro that takes a path through any of the places identified as the beginning of macro servicing can be intercepted. It is, however, assumed that the user will screen out those macros issued from the control program.

The macros which generate inline code are not capable of being intercepted at macro entry or at macro exit because there are no exit points at these locations. The WGTAC and CINFC (fast path) macros are inline macros and, therefore, are not intercepted.

There are some macro service routines that perform a control transfer operation as part of their processing. If a core block cannot be obtained for the control transfer, the macro service routine may temporarily suspend the ECB by directly invoking the DLAYC macro processing routine. If this occurs, the User Exit routine will be invoked next at postinterrupt time for the DLAYC macro (if the SVC macro exit with postinterrupt exit point SVW is active), and then again when the original macro service routine completes its processing (if the associated exit point is active).

Note: Some macros that issue an implied wait will return via the SVX exit point if no outstanding I/O exists. If outstanding I/O exists, the macro will return using SVW.

Control Program User Exits

This chapter describes each TPF control program (CP) user exit, how to use the user exit, and the following information concerning each user exit:

- General conditions at entry
- System conditions at entry
- Programming considerations at entry
- Programming considerations on return
- General conditions on return.

Application Timeout Processing

User exit routine UCCAPL provides the ability to perform additional actions or extend the life of an ECB that is about to be timed out.

UCCAPL is called when the ECB that is currently running exited with a 000010 or a 00002010 system error. Exit point APL is located in 2 places in copy segment CTME of CSECT CCNUCL; just prior to issuing the 000010 system error and just prior to issuing the 00002010 system error.

General Conditions at Entry

The registers at entry to UCCAPL are:

R0	Set to 0.
R8	Address of the program's PAT slot.
R9	SVM address of the ECB.
R11–R15	See "User Exit Routines - Common Entry Conditions" on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O and external interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

- See "User Exit Routines - Common Entry Conditions" on page 6.
- See "User Exit Routines - Common Programming Considerations" on page 7 for other considerations.
- PFXATMR will be zero if a 000010 system error is going to be issued. Otherwise, a 00002010 system error will be issued and CE2TMSLC contains the address of the time-slice name entry in the time-slice name table.
- Calling TPF services from within this user exit is not supported.

Programming Considerations on Return

For all registers except R0, see "User Exit Routines - Common Return Conditions" on page 7.

General Conditions on Return

- R0 will be set to 0 if a 000010 or 00002010 dump will be taken. A nonzero return code in R0 will result in no dump being taken and control returns to the application program. If user exit was called on behalf of 000010 system error processing, the value of R0 will be used to reset PFXATMR on return from the user exit.
- The user exit code sets any fields (for example, CE1ISTIM and PFXATMR) to prevent additional errors or timeouts.
- See "User Exit Routines - Common Return Conditions" on page 7.

BACKC Macro Entry

User exit routine UCCCFB is called at the beginning of macro processing for a BACKC macro (if the exit point named CFB is active). Exit point CFB is in CSECT CCENBK.

General Conditions at Entry

The registers at entry to UCCCFB are:

R8	Base of E-type program that issued the macro.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 3)
System mask	Unmasked (see “Programming Considerations at Entry”, item 3)
Protect key	Working storage or zero (0)
Address Space	EVM.

Programming Considerations at Entry

1. The contents of program registers R0–R7 have not been saved in the ECB.
2. Information on the program to return to can be found in the current program nesting level, CE1CPNL.
3. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state and changed the system mask.
4. The CFB exit point occurs before formatting an entry in the macro trace table.
5. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

C Debugger Initialization

User exit routine UCCCDBI allows you to initialize a C debugger other than the TPF C Debugger for VisualAge Client. UCCCDBI is called before initializing a C debugger if the exit point named CDBI is active. Exit point CDBI is located in the object code only segment CDBINT, which is part of the library load module CISO.

General Conditions at Entry

The registers at entry to UCCCDBI are:

- R0–R8, R10** Contents are unknown.
- R9** Address of the ECB.
- R11–R15** See “User Exit Routines - Common Entry Conditions” on page 6.

The C debugger state indicators in page 2 of the ECB are:

CE2DBENBL Indicates if the ECB is enabled for debugging. If it is, one of the following fields in the TPF process block (IPROC) will contain a pointer to the trace entry; the other will contain a zero to indicate that it is not being used:

IPROC_TBT_PTR

The address of the trace-by-terminal entry.

IPROC_TBP_PTR

The address of the trace-by-program entry.

Each trace entry contains the following information:

- Debug workstation IP address
- Port number of the debug workstation
- Entry state flag
- Option flags
- Number of ECBs using this entry
- Created ECB trace token
- Programs to be traced
- User token.

Note: A trace-by-terminal entry also contains a field that identifies the terminal information. A terminal can be a TCP/IP address, a fully qualified LU Name, or an LNIATA.

CE2DBHOOK Indicates if the ECB has entered a program that matches the trace entry information and, therefore, should initialize the debugger.

CE2DBINIT Indicates if the debugger has been initialized.

CE2DBFLAG This byte holds several internal flags that are used by debuggers. One of the bits indicate that the debugger has issued an input/output (I/O) command to the workstation and is waiting for a response. This bit is set before writing to the socket and is cleared after reading the socket.

System Conditions at Entry

System state

Inherited from application

System mask	Inherited from application
Protect key	Inherited from application
Address Space	EVM.

Programming Considerations at Entry

1. UCCCDBI can use the data stored in the user token field of the trace entry, or any other information accessible by the ECB to determine if it should delay the initialization of the debugger until the ECB enters another C program. To delay the initialization of the debugger until the ECB enters another C program, set the CE2DBINIT field to CE2DBINIT_OFF.
2. See “User Exit Routines - Common Entry Conditions” on page 6.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

C Debugger Return

User exit routine UCCCDDBR is called if the exit point named CDBR is active. UCCCDDBR is called from the TPF C Debugger for VisualAge Client routine that handles the debugger hooks. The exit point is given control prior to returning to the application. Exit point CDBR is located in CPLX library load module.

General Conditions at Entry

R0, R2–R8, R10

Contents are unknown.

R1

Parameter list for user exit containing the following information:

- Application program mask and condition code (full word)
- Address in application where debugger will return control
- Application registers (R0–R15)

R9

Address of the ECB.

R11–R15

See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Inherited from application
System mask	Inherited from application
Protect key	Inherited from application
Address Space	EVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

C Library Function Call (TARGET(TPF))

User exit routine UCCCFC is used with C language support and is activated on a call to a C library function if the exit point named CFC is active. Exit point CFC is in CSECT CCLANG.

General Conditions at Entry

The registers at entry to UCCCFC are:

R7	C stack pointer.
R8	Index into the quick enter table.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working Storage
Address state	EVM.

Programming Considerations at Entry

1. It is possible to be in Protect key 0 if a CINFC(CINFC_WRITE) was issued.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

C Library Function Return (TARGET(TPF))

User exit routine UCCCFR is used with C language support and is activated on return from a C library function if the exit point named CFR is active. Exit point CFR is in CSECT CCLANG.

General Conditions at Entry

The registers at entry to UCCCFR are:

R7	C stack pointer.
R8	Base address of C program to return to.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working Storage
Address state	EVM.

Programming Considerations at Entry

1. It is possible to be in Protect key 0, if a CINFC(CINFC_WRITE) was issued. The protect key can be different from the key at UCCCFR.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

C Stack Exception (TARGET(TPF))

The user exit (UCCCSK) for the C stack exception routine is used with C language support and is activated the first time a C program is called during the life of an entry.

Some possible uses of this exit point include:

- To specify a locale other than the default locale for the current ECB
- To add a user expansion area at the end of the first stack frame
- To collect utilization/performance statistics
- To use for your own tracing or debugging tools.

Exit point CSK in CSECT CCLANG. When the user exit is activated, control is transferred to the CUSR copy member of CCUEXT.

General Conditions at Entry

The registers at entry to UCCCSK are:

R1	Current block pointer.
R4	Address of next C stack frame (CSTKNAB).
R5	Address of default locale name.
R6	Requested C stack frame size.
R7	Address of first C stack frame in first block.
R8	Program base register.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Unmasked
Protect key	0
Address space	EVM.

Programming Considerations at Entry

1. If you want to add additional space at the end of the first stack frame, this space must be added immediately following the CSTKEND field. The length of this additional space must be added to both CSTKNAB and R4 (the new frame pointer), which hold the same value. The maximum size of the stack frame is 4095 bytes, minus the size of the block header. For detailed information about the layout of the stack frame, see data macro ICS0TK.
2. The current locale name (the default) is pointed to by R5. If a different locale is needed, store the name of the locale (4-byte maximum) at the location pointed to by R5. Characters EDC\$ at the beginning of each locale name are not stored. For example, for locale EDC\$UK, store UK at the address pointed to by R5, and pad this value on the right with nulls.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

All registers must be restored before return, with the following exception. If the size of the first stack frame is expanded because of the addition of a user expansion area, then R4 is updated to point to the next stack frame.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7, except for R4, as discussed previously.

C Stack Exception Return (TARGET(TPF))

User exit routine UCCCSER is used with C language support and is activated on return from a stack or static exception if the exit point named CSER is active. Exit point CSER is in CSECT CCLANG.

General Conditions at Entry

The registers at entry to UCCCSER are:

R9 Address of the ECB.

R13–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working storage
Address state	EVM.

Programming Considerations at Entry

1. It is possible to be in Protect key 0 if a CINFC(CINFC_WRITE) was issued.
2. The contents of the caller's registers upon entry to the user exit routine, can be retrieved from the stack in R13 by entering the following in the user exit:

```
L R5,STKPREV(,R13)
L R3,STKR0(R5)                      THIS GETS R0
```

These entries must be made before the SLNKC macro statement at the beginning of the exit routine.

3. This exit will be activated by a STATIC or STACK exception. It is the user's responsibility to determine which occurred.
4. Some registers saved in the stack have been modified by the exception handlers. See a listing of CCLANG for details.
5. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Catastrophic Recovery

User exit routine UCCCAT is invoked from catastrophic recovery processing after the tape queues have been cleared, but before tapemarks are written to the real-time tapes. Exit point CAT is in CSECT CCCPSF.

UCCCAT is provided to give users the opportunity to log critical records to tape. UCCCAT is invoked once for every active real-time tape found in the real-time tape chain. UCCCAT should first examine the tape status table entry to see if it is a tape for which user exit processing is required. If it is, UCCCAT should invoke CEDTWBK, in the CSECT CCCPSE, to write additional blocks to tape.

Note: Save your registers and change them as needed before you call CEDTWBK from the UCCCAT user exit.

Routine CEDTWBK expects the following conditions at entry:

R0	The length of the record to be written. Length must not exceed 32 690 bytes.
R1	The main I-stream system virtual address of the record to be written.
R2	Module number of the tape being written to (supplied by CCCPSF on entry to the user exit).

General Conditions at Entry

The registers at entry to UCCCAT are:

R0	Module number of the tape being processed. At the conclusion of user exit processing, CCCPSF will send the dummy record/tapemark sequence to the referenced tape device.
R6	Tape status table section 1 entry for the tape being processed. This register can be used to determine whether the tape name in the tape status table entry identifies a tape that requires user processing. See the ITSTB DSECT.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

Refer to a listing of CCCPSF for the contents of the remaining registers.

System Conditions at Entry

System state	Supervisor, normal I/O suspended
System mask	Masked
Protect key	0
Address space	SVM on the main I-stream.

Programming Considerations at Entry

1. User processing must not change the system state. Return must be made to the next sequential instruction.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

CCCTIN (CT25 and CT26)

The CTIN user exits are invoked in copy segment CT00 of CSECT CCCTIN. They are nondynamic exit points that do not follow the standards set by the other exit points. The exit routines are not in CUSR. Instead, they have their own copy segments, CT25, CT26, and CT99 in CCCTIN. The user exits control list (UCL) is not used, nor are the linkage macros used. This is because during initialization, many control program tables and system stacks are not yet available.

The CCCTIN exit permits the user to reserve and initialize main storage for user definition.

To activate exit points CT25 and CT99, you must change the statement at the beginning of each copy segment from **EQU INIT00** to **DS F**.

CT00 tests for the relative location of CT25 and CT99. If the result is 0, the exit points are bypassed. CT26 is activated only when CT25 is in use. Refer to the commentary in the program listing for copy members CT25, CT26, and CT99 for additional information.

General Conditions at Entry

The registers at entry to CT25 are:

R1	Set to 0
R4	Address of next available main storage location
R8	Base of CTIN
R9	Base of CT25
R14	Return address.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	IPLVM.

Programming Considerations at Entry

1. On return, R1 must either be zero or contain the address of the user storage allocation table (USAT). If an address is returned, CT26 is activated to allocate storage for the user using the information in the USAT. Macro IUSAT is used to describe the layout of each individual entry in the USAT. The maximum number of entries is defined by the variable &SAADLN in SYSET. As released by IBM, this value is 50.
2. CT25 is provided to allow you to reserve storage for user tables or other user-unique structures that require reserved main storage. This can be done through the following methods:
 - a. Build a user storage allocation table, USAT, and pass the address to CT26, which does the actual allocation. The USAT should be built, using the USATC macro and assembled into CT25.
 - b. Point to a USAT built elsewhere, perhaps in a user keypoint, and pass the address to CT26.

- c. Allocate storage directly in CT25 by updating R4, the next available storage pointer. However, this is not recommended.

CT25 can also be used to relocate the user data in the FCTB. The core address of the FCTB can be obtained using the CINFC tag, CMMFCTV.

Programming Considerations on Return

R1	Address of USAT or 0.
R4	Must be unchanged or, if main storage was reserved by CT25 and the next available location has changed, R4 must contain the address of the next available main storage location rounded to the next doubleword.
R8	Must still contain the base of CTIN.

General Conditions on Return

The System Mask, System State, and Protect Key must be the same as on entry to CT25.

CCCTIN (CT99)

The CTIN user exits are invoked in copy segment CT00 of CSECT CCCTIN. They are nondynamic exit points that do not follow the standards set by the other exit points. The exit routines are not in CUSR. Instead, they have their own copy segments, CT25, CT26, and CT99 in CCCTIN. The user exits control list (UCL) is not used, nor are the linkage macros used. This is because during initialization, many control program tables and system stacks are not yet available.

The CCCTIN exit permits the user to reserve and initialize main storage for user definition.

To activate exit points CT25 and CT99, you must change the statement at the beginning of each copy segment from **EQU INIT00** to **DS F**.

CT00 tests for the relative location of CT25 and CT99. If the result is 0, the exit points are bypassed. CT26 is activated only when CT25 is in use. Refer to the commentary in the program listing for copy members CT25, CT26, and CT99 for additional information.

General Conditions at Entry

The registers at entry to CT99 are:

R8	Base of CTIN
R9	Base of CT99
R14	Return address.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

CT99 is provided to allow users to initialize/key protect user unique main storage tables, allocated by CT25 and CT26, or other user-unique structures, such as user CINFC labels.

Programming Considerations on Return

R8	Must still contain the base of CTIN.
-----------	--------------------------------------

General Conditions on Return

The System Mask, System State, and Protect Key must be the same as on entry to CT99.

Control Transfer

User exit routine UCCCMXF is invoked whenever a control transfer macro (CXFRC) is executed and the exit point named CMXF is active. Exit point CMXF is contained in CSECT CCNUCL. Exit point CMXF contains different parameters depending on the CXFRC ECB= parameter.

UCCCMXF allows user information to be passed from a parent to a child ECB. It is independent of macro servicing user exits.

General Conditions at Entry

The registers at entry to UCCCMXF are:

R2	Pointer to a 4-byte user data field where this routine can store data to be passed to the child ECB. R2 equals 0 if CXFRC was unable to obtain storage because of a low core condition.
R5	Set to 1 to show that ECB=DEFER was coded.
R7	Return address of CXFRC (CXFRTRN).
R9	If present, the address of the parent ECB. See "Programming Considerations at Entry".
R11–R15	See "User Exit Routines - Common Entry Conditions" on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	SVM or EVM.

General Conditions at Entry

The registers at entry to UCCCMXF are:

R2	Address of the new ECB.
R5	Set to 0 to show that ECB=IMMED was coded.
R7	Return address of CXFRC (CXFRTRN).
R9	If present, the address of the parent ECB. See "Programming Considerations at Entry".
R11–R15	See "User Exit Routines - Common Entry Conditions" on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. A parent ECB may or may not be present.
2. See "User Exit Routines - Common Entry Conditions" on page 6.

3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Core Resident Enter/Back Macro

User exit routine UCCCREB is called if the exit point named CREB is active. UCCCREB is invoked at the end of macro processing and just before control is passed to the destination program for:

- Enter-type macros going to a core-resident program
- BACKC macro from a core-resident program.

UCCCREB can be called from an ISO-C environment. Exit point CREB is located in copy segment CCEB of CSECT CCENBK and in copy segment CLHL of CSECT CCCLHR.

Multifunction user exit linkage is assembled into this exit point. This user exit is activated by the ZDEBUG command (with the START parameter specified), which calls the UXITC macro with the IUX_VADB function set on.

General Conditions at Entry

The registers at entry to UCCCREB are:

R0–R7	When the user exit is called from a non-ISO-C environment, the contents have been restored from the ECB and contain the values that are to be passed to the E-type program that will receive control. When the user exit is called from an ISO-C environment, the application registers are stored in the data save area (DSA) and have not yet been restored.
R8	Base of E-type program to receive control.
R9	Address of the ECB.
R10	Contains 0 if the user exit is called from an ISO-C environment, otherwise it contains a non-zero value.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage.
Address Space	EVM.

Programming Considerations at Entry

1. Usually, the system state, system mask, and protection key are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state, changed the system mask, or changed the protection key.
2. You activate the TPF Assembler Debugger for VisualAge Client or TPF C Debugger for VisualAge Client code from this user exit by calling the UXMAC macro. Program load event detection code (the IUX_VADB UXMAC call) must be the **last** exit routine to be called in the exit. The debugger will **not** return to this user exit when it determines that the debugger must receive control. If you do not want the debugger to trace the program that will be entered, set the

CE3_NO_DEBUG indicator in byte CE3DBCR (page 3 of the ECB) before the debugger is called. The debugger always sets this indicator off before activating the program.

3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

CPU External Interrupt

User exit routine UCCCPTE is invoked whenever any CPU external interrupt is taken and the exit point named CPTE is active. Exit point CPTE is contained in CSECT CCNUCL.

This exit point has been provided to allow the user exit routine to be invoked when any CPU external interrupt is received. The exit will be invoked from both the main and application I-streams.

General Conditions at Entry

The registers at entry to UCCCPTE are:

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

CPU Timer Interrupt

User exit routine UCCCPTI is invoked whenever a CPU timer clock interrupt is taken and the exit point named CPTI is active. Exit point CPTI is contained in CSECT CCNUCL.

This exit point has been provided to allow the user exit routine to be invoked when an external CPU timer interrupt is received. The exit will be invoked from both the main and application I-streams.

General Conditions at Entry

The registers at entry to UCCCPTI are:

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Create Macro Control Point

User exit routine UCCCMCP is called whenever a create macro (CREDC, CREEC, CREMC, CRESC, CRETC, CREXC, or SWISC) is executed and the exit point named CMCP is active. Exit point CMCP is contained in CSECT CCNUCL.

UCCCMCP allows user information to be passed from a parent to a child entry control block (ECB). It is independent of macro servicing user exits. For calls from the \$FORKC macro, passing information from a parent to a child ECB must be done in UCCCMCP because the UCCCMPI user exit will not be invoked.

General Conditions at Entry

The registers at entry to UCCCMCP are:

R2 Pointer to a 4-byte user data field.

R6

For calls from the \$FORKC macro, R6 contains the address of the child ECB. For other calls, the content of R6 is undefined.

R9 Address of the ECB that issued the macro.

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state Supervisor

System mask Masked for I/O interrupts

Protect key 0

Address Space

EVM, except for calls from the \$FORKC macro in which case the address space is in SVM.

Programming Considerations at Entry

1. Except for calls from \$FORKC, the 4-byte user data field is initialized to zero when UCCCMCP is entered. For calls from the \$FORKC macro, the 4-byte user data field is initialized to 3 and must not be changed.
2. If you perform CRETCM processing, the 4-byte user data field is set to 1. If you perform CRETCS processing, the 4-byte user data field is set to 2. It is your responsibility to reset the field to its initial value.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Create Macro Postinterrupt

User exit routine UCCCMPI is invoked whenever a new entry control block (ECB) is obtained because of Create postinterrupt processing and the exit point named CMPI is active. Exit point CMPI is contained in CSECT CCNUCL.

UCCCMPI allows user information to be passed from a parent to a child ECB. It is independent of macro servicing user exits.

Note: For calls from the \$FORKC macro, passing information from a parent to a child ECB must be done in UCCCMCP because the UCCCMPI user exit will not be invoked.

General Conditions at Entry

The registers at entry to UCCCMPI are:

R2	Pointer to a 4-byte user data field.
R9	Address of the ECB that was just created.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Critical Record Filing

User exit routine UCCCRI is invoked during critical record filing of catastrophic error processing. This exit point **cannot** be activated dynamically. Exit point CRI is in copy segment CPSF of CSECT CCCPSF.

General Conditions at Entry

The registers at entry to UCCCRI are:

R0–R8	Contents unknown.
R9	Address of the ECB.
R10	CPSF base address.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for all interrupts
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Debugger Trace Selection

User exit routine UCCDBTS is called when the TPF Assembler Debugger for VisualAge Client is about to assign a trace-by-program entry to the application entry control block (ECB). This user exit permits you to verify additional information in the ECB against a user token field that is stored in the trace entry. You supply the user token data on the TPF registration window. UCCDBTS is called if the DBTS exit point is active. Exit point DBTS is located in CSECT CCENBK.

General Conditions at Entry

The registers at entry to UCCDBTS are:

R1	I PROG trace entry that will be assigned to the ECB.
R6	Pointer to page 2 of the ECB.
R7	Pointer to the name of the program that will be entered.
R9	Pointer to page 1 of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Inherited from the application
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7 for considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. Register 5 (R5) must contain a return code.
 - 0** Returning a value of 0 causes the trace entry to be attached to the ECB.
 - 1** If you decide that the trace entry that was passed to the user exit should not be attached to the ECB, return a value of 1. Segment CCED continues to search its tables for another trace entry that matches this ECB. The user exit is called again if another match is found.
2. See “User Exit Routines - Common Return Conditions” on page 7.

Debugger Trace Table Entry Activation

User exit routine UCCDBTA is called when the TPF Assembler Debugger for VisualAge Client is about to activate a trace-by-program or trace-by-terminal entry. This user exit permits you to perform additional customization or initialization before the trace session starts. UCCDBTA is called if the DBTA exit point is active. Exit point DBTA is located in CSECT CCENBK.

General Conditions at Entry

The registers at entry to UCCDBTA are:

- | | |
|----------------|---|
| R9 | Pointer to page 1 of the ECB. |
| R11–R15 | See “User Exit Routines - Common Entry Conditions” on page 6. |

System Conditions at Entry

System state	Supervisor
System mask	Inherited from the application
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. Access the debugger trace process block (IPROC) that is pointed to by the addresses in CE2PROC to determine which trace table is being used. The address of the trace table entry is in one of the following:
 - IPROC_TBT_PTR, which is defined by ITERM
 - IPROC_TBP_PTR, which is defined by IPROG.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

DLAYC Macro Entry

User exit routine UCCDLAY is called during macro processing for an DLAYC or YIELDC macro (if the exit point named DLAY is active). The exit point is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry are:

R6	Type of macro calling this exit point.
01	DLAYC macro call
02	YIELDC macro call (for the READY list)
02	YIELDC macro call (for the virtual file access count (VCT) list)
R9	Pointer to entry control block (ECB) page 1.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	Working storage or zero (0)
Address Space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

- Intermediate processing contains code that performs special processing for virtual machine (VM) and processor resource/systems manager (PR/SM) environments. This processing can load a wait state program status word (PSW) under certain conditions; load RC=4 to register 6 (R6) and return to the caller if you want to bypass intermediate processing.
- DLAYC macro processing does not check to see if the ECB is holding any resources. With the DLAYC macro entry user exit, you can determine whether the ECB is holding any resources. Additionally, you can decide to issue a system error or return to the issuing program without entering the DLAYC macro. To ignore the delay request, load RC=8 to R6 and return to the service routine. The service routine ignores the delay request and returns to the program that issued the macro.
- See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

- R6 must contain the return code that will be used by the calling service routine to determine the next action.
 - 0** Continue normal processing.
 - 4** Bypass intermediate processing and go directly to delay processing.
 - 8** End the delay macro call and return to the macro caller.

- See “User Exit Routines - Common Return Conditions” on page 7.

Dump Override Table

User exit routine UCCDOT is invoked from system error processing after the selective memory dump table has been initialized for a dump of system storage. Exit point DOT is located in CSECT CCCPSE. This exit is provided to give users final control over which storage areas are included in the dump.

General Conditions at Entry

The registers at entry to UCCDOT are:

- R2** Base address of the selective memory dump table. The SMDT resides in the copy member of the CCCPSE CSECT.
- R11–R15** See “User Exit Routines - Common Entry Conditions” on page 6.

Note: Refer to a listing of CCCPSE for the contents of the remaining registers.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM on the main I-stream.

Programming Considerations at Entry

1. User processing must not change the system state. Return must be made to the next sequential instruction.
2. The format of an SMDT entry is described by the IDSMDT data macro. To obtain the address of the SMDT entry corresponding to a particular keyword, use the following code:

```
IDSMDT REG=Rx          DSECT AN SMDT ENTRY
LA Rx,<keyword>        LOAD KEYWORD EQUATE
MH Rx,=Y(MDTLEN)       TIMES SIZE OF ENTRY
A Rx,0(,R2)            ADD TABLE BASE
```

To flag an SMDT entry for inclusion in the dump, set the MDTINCL flag in the MDTFLG flag byte. Also, you can omit storage areas from the dump by turning off MDTINCL.

3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Dynamic Load Module Environment Initialization (ISO-C)

User exit routine UCCCENV is used with ISO-C support. UCCCENV is activated when an ISO-C environment is initially created for a given ECB after all ISO-C control structures have been initialized, if the exit point named CENV is active. User exit CENV is called from an ISO-C environment. Exit point CENV is called from the dynamic load module (DLM) startup code (CSTRTD) of the CLMINT segment in the CIS0 library.

Some possible uses of this exit point include:

- To increase the size of the language work space (LWS)
- To load a default locale that is different than the C/370 locale.

General Conditions at Entry

The registers at entry to UCCCENV are:

R5	0.
R8	Points to the DLM header.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

The relevant ECB fields at entry to UCCCENV are:

CE2ISOC	Address of the Task Communication Area (TCA)
CE3SPTR	Address of the Data Save Area (DSA).

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 2)
System mask	Unmasked (see “Programming Considerations at Entry”, item 2)
Protect key	Working storage
Address state	EVM.

Programming Considerations at Entry

1. The language work space (LWS) area cannot be incremented past the end of the stack (EOS) field in the task communication area (TCA).
2. Usually, the system state, system mask, and protection key are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state, changed the system mask, or changed the protection key.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. If you need a locale that is different from the default locale, use R5 to indicate the locale. For EDCLOC-based locales, R5 must contain the name of the

module that contains the locale. For `localedef` utility-based locales, R5 must contain the 4-character internal name of the locale.

2. See “User Exit Routines - Common Return Conditions” on page 7.

Dynamic Load Module External Function Call

User exit routine UCCEFCX is called by the dynamic load module (DLM) startup code before calling the entry point function (if the exit point named EFCX is active). User exit EFCX is called from an ISO-C environment. This exit point is in the CLMINT segment of the CIS0 library.

Multifunction user exit linkage is assembled into this exit point. This user exit is activated by the ZDEBUG command (with the START parameter specified), which calls the UXITC macro with the IUX_VADB function set on.

General Conditions at Entry

The registers at entry are:

R8	Program base of the called program
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage or zero (0)
Address Space	EVM.

Programming Considerations at Entry

1. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued a MONTC macro to get to supervisor state and changed the system mask.
2. You activate the TPF Assembler Debugger for VisualAge Client or TPF C Debugger for VisualAge Client code from this user exit by calling the UXMAC macro. Program load event detection code (the IUX_VADB UXMAC call) must be the **last** exit routine to be called in the exit. The debugger will **not** return to this user exit when it determines that the debugger must receive control. If you do not want the debugger to trace the program that will be entered, set the CE3_NO_DEBUG indicator in byte CE3DBCR (page 3 of the ECB) before the debugger is called. The debugger always sets this indicator off before activating the program.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Dynamic Load Module External Function Call Entry

User exit routine UCCEFCE is called during the processing of an external function call by a dynamic load module (if the exit point named EFCE is active). This exit point is in CSECT CCENBK (CCED).

General Conditions at Entry

The registers at entry to UCCEFCE are:

R8 Base of E-type program (dynamic load module) that issued the external function call.

R9 Address of the ECB.

R11, R12, R13, R15

See "User Exit Routines - Common Entry Conditions" on page 6.

Relevant ECB fields at entry are:

CE3SPTR Data save area (DSA) pointer.

System Conditions at Entry

System state Problem (see "Programming Considerations at Entry", item 2)

System mask Unmasked (see "Programming Considerations at Entry", item 2)

Protect key Working storage or zero (0)

Address Space EVM.

Programming Considerations at Entry

1. On entry, the ECB field CE3PAT will contain the PAT entry address for the program (dynamic load module) that issued the external function call.
2. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued a MONTTC macro to get to supervisor state and changed the system mask.
3. The dynamic load module external function call exit point occurs before the formatting of an entry in the macro trace table.
4. See "User Exit Routines - Common Programming Considerations" on page 7 for other considerations.

Programming Considerations on Return

See "User Exit Routines - Common Return Conditions" on page 7.

General Conditions on Return

See "User Exit Routines - Common Return Conditions" on page 7.

Dynamic Load Module Return Processing

User exit routine UCCRTNX is called if the exit point named RTNX is active. UCCRTNX is called at the end of return processing from a dynamic load module (DLM). User exit RTNX may be called from an ISO-C environment. Exit point RTNX is in CSECT CCENBK (CCED).

Multifunction user exit linkage is assembled into this exit point. This user exit is activated by the ZDEBUG command (with the START parameter specified), which calls the UXITC macro with the IUX_VADB function set on.

General Conditions at Entry

The registers at entry to UCCRTNX are:

R0–R7	When the user exit is called from a non–ISO-C environment, the contents have been restored from the ECB and contain the values that are to be passed to the E-type program that will receive control. When the user exit is called from an ISO-C environment, the application registers are stored in the data save area (DSA) and have not yet been restored.
R8	Base of E-type program to receive control.
R9	Address of the ECB.
R10	Contains 0 if the user exit is called from an ISO-C environment, otherwise it contains a non-zero value.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage
Address Space	EVM.

Programming Considerations at Entry

1. Usually, the system state, system mask, and protection key are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state, changed the system mask, or changed the protection key.
2. You activate the TPF Assembler Debugger for VisualAge Client or TPF C Debugger for VisualAge Client code from this user exit by calling the UXMAC macro. Program load event detection code (the IUX_VADB UXMAC call) must be the **last** exit routine to be called in the exit. The debugger will **not** return to this user exit when it determines that the debugger must receive control. If you do not want the debugger to trace the program that will be entered, set the CE3_NO_DEBUG indicator in byte CE3DBCR (page 3 of the ECB) before the debugger is called. The debugger always sets this indicator off before activating the program.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Dynamic Load Module Return Processing Entry

User exit routine UCCRTNE is called at the beginning of return processing for a dynamic load module (if the exit point named RTNE is active). User exit RTNE is called from an ISO-C environment. Exit point RTNE is in the CLMINT segment of the CISO library.

General Conditions at Entry

The registers at entry to UCCRTNE are:

R8	Base of E-type program (dynamic load module) that is returning.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 2)
System mask	Unmasked (see “Programming Considerations at Entry”, item 2)
Protect key	Working storage or zero (0)
Address Space	EVM.

Programming Considerations at Entry

1. For information on the program in which to return, see the current program nesting level, CE1CPNL.
2. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state and changed the system mask.
3. The RTNE exit point occurs before formatting an entry in the macro trace table.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

ECB Creation

User exit routine UCCECB is invoked whenever a new entry control block (ECB) is created, if the exit point named ECB is active. Exit point ECB is located in the ECB creation routine contained in CSECT CCNUCL (CHSZ). UCCECB provides the ability to examine the origin (via the ECB format flag) of all ECBs and to subsequently track individual ECBs.

General Conditions at Entry

The registers at entry to UCCECB are:

R0–R8	The contents are unknown because the ECB exit point is in the ECB creation subroutine, and therefore the contents of these registers are dependent on the calling routines.
R9	Address of the ECB that was just created.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. Because the ECB exit point is in a subroutine called from more than 1 point in the control program, user processing must ***not give up control, and must return to the NSI*** contained in register 14. The ECB cannot be exited and control cannot be passed to the CPU loop (Task Dispatcher).
2. At entry to UCCECB, the ECB has been allocated and initialized, including the ECB format flag at label CE1FLG in the ECB. The format flag value can be used to determine the type of ECB (for example, control transfer, SNA input message, and others).
3. At entry to UCCECB, the process control block associated with the ECB has been allocated. The default group ID and user ID (IPROC_GID and IPROC_UID) can be modified at this point. The group ID and user ID are used by the file system to handle file permissions. You can find the address of the process control block at CE2PROC in page 2 of the ECB. The IPROC DSECT maps the process control block fields.
4. The environment variables can be set to specify the initial current working directory, the standard input (stdin) file, the standard output (stdout) file, and the standard error (stderr) file.
5. The following fields in the ECB can be modified:
 - Maximum amount of storage for the C language stack (CE2MHSF)
 - Maximum amount of storage for TPF heap storage (CE2MPF)
 - Initial stack allocation (CE2ISAS)
 - Stack increment (CE2SISZ).
6. See “User Exit Routines - Common Entry Conditions” on page 6.
7. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

ENTxC Macro Entry

User exit routines UCCENTR, UCCENTN, and UCCENTD are called during macro processing for an ENTRC, ENTNC, or ENTDC macro (if the exit points named ENTR, ENTN, or ENTD are active). These exit points are in CSECT CCENBK.

General Conditions at Entry

The registers at entry are:

R8	Base of E-type program that issued the macro, unless ENTNC was issued from CP code
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 5)
System mask	Unmasked (see “Programming Considerations at Entry”, item 5)
Protect key	Working storage or zero (0)
Address Space	EVM or SVM.

Programming Considerations at Entry

1. The contents of program registers R0–R7 have not been saved in the ECB register save area.
2. On entry, the ECB field CE3PAT will contain the PAT entry address for the program that issued the macro, unless the macro was issued from CP code, in which case CE3PAT will be zero. When CE3PAT is zero, R8 cannot be predicted.
3. Entry is in the SVM when the ENTNC is issued from CP code.
4. Access to the ENTxC macro parameters can be found by retrieving the pointer address from stack field STKINL14(R13).
5. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued a MONTC macro to get to supervisor state and changed the system mask.
6. The ENTxC exit points occur before the formatting of an entry in the macro trace table.
7. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

EXITC

User exit routine UCCEXI is called from the EXITC macro service routine if the exit point named EXI is active. Exit point EXI is located in copy segment CCEB of CSECT CCENBK.

UCCEXI provides the ability to trap all ECB exits. It is independent of the macro service user exit.

General Conditions at Entry

The registers at entry to UCCEXI are:

R0–R8	Contents unknown. The registers have been restored from the ECB register save area.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

Performance-related ECB fields filled in:

CE1CTRS	the label for ECB performance counters area
CE1DSTMP	The 8-byte TOD clock time of most recent dispatch.
CE1EXTIM	The 8-byte TOD clock time of ECB exit. Set immediately before the EXI exit point.
CE1ISTIM	The 8-byte TOD clock format of accumulated running time.
CE1FINDS	The 4-byte count of Find requests (counts FINDC, FINHC, FINSC, FINWC, FIWHC, and FNSPC macros).
CE1FILES	The 4-byte count of File requests (counts FILEC, FILNC, FILSC, FILUC, and FLSPC macros).
CE1GETFCS	The 4-byte count of GETFC requests.
CE1USERID	The 4-byte field for user information initialized to zero at creation and not modified by the TPF system.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address space	EVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

FARF Address Generation

If the exit point named FSP is active then user exit routine UCCFSP is invoked during FACE or FACS processing after the FARF address for the record being requested has been generated. Exit point FSP is located in copy segment CUSR of CSECT CCUEXT.

General Conditions at Entry

The registers at entry to UCCFSP are:

R0	Input ordinal number.
R4	Pointer to split for requested record.
R6	Pointer to split chain header for requested record type.
R7	Output area containing FARF address.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	1
Address space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

Programming Considerations on Return

The following registers can be modified on return:

R0	Input ordinal number.
R6	Pointer to split chain header for requested record type.

Note: R13 must contain a return code set by the user exit routine to indicate the action to be taken. The following return codes are supported:

- 0 Continue normal processing.
- 4 The contents of R0, R6, or both have been modified; retry the FACE or FACS call.
- 8 Return an error indication to the FACE or FACS call.

See “User Exit Routines - Common Return Conditions” on page 7 for additional return conditions.

General Conditions on Return

User processing must return to the NSI. For more programming considerations, see “User Exit Routines - Common Return Conditions” on page 7.

Fast Link Macro

Exit routine UCCFLX is called at the end of macro processing for fast link macros, just before returning control to the E-type program that issued the fast link macro (if the exit point named FLX is active). Exit point FLX is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCFLX are:

R0–R7	The contents cannot be predicted. The values being returned to the caller are in the ECB register save area. If the fast link macro returns a condition code, bits 2 and 3 of R0 contain the condition code that is being returned.
R8	Base of E-type program that issued the fast link macro.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage.
Address Space	EVM.

Programming Considerations at Entry

1. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state and changed the system mask.
2. See “User Exit Routines - Common Entry Conditions” on page 6.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Fast Link Macro Decoder

User exit routine UCCFLM is invoked from the fast link macro decoder if the FLM exit point is active. Exit point FLM is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCFLM are:

R6	Address of the macro entry in the macro decoder table.
R8	Base of E-type program that issued the fast link macro.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 2)
System mask	Unmasked (see “Programming Considerations at Entry”, item 2)
Protect key	Working storage
Address Space	EVM.

Programming Considerations at Entry

1. The contents of program registers R0–R7 have been saved in the ECB register save area.
2. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state and changed the system conditions.
3. This exit includes ENTRC to FACE or FACS.
4. The FLM exit point occurs before formatting an entry in the macro trace table.
5. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

File-Resident Enter/Back Macro

User exit routine UCCFREB is called (if the FREB exit point is active) at the end of macro processing and just before control is passed to the destination program for:

- Enter-type macros going to a file-resident program
- The BACKC macro from a file-resident program.

UCCFREB can be called from an ISO-C environment. Exit point FREB is in copy segment CCEB of CSECT CCENBK and in copy segment CLHL of CSECT CCCLHR.

Multifunction user exit linkage is assembled into this exit point. This user exit is activated by the ZDEBUG command (with the START parameter specified), which calls the UXITC macro with the IUX_VADB function set on.

General Conditions at Entry

The registers at entry to UCCFREB are:

R0–R7	When the user exit is called from a non-ISO-C environment, the contents have been restored from the ECB and contain the values that are to be passed to the E-type program that will receive control. When the user exit is called from an ISO-C environment, the application registers are stored in the data save area (DSA) and have not yet been restored.
R8	Base of E-type program to receive control.
R9	Address of the ECB.
R10	Contains 0 if the user exit is called from an ISO-C environment, otherwise it contains a non-zero value.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage
Address Space	EVM.

Programming Considerations at Entry

1. Usually the system mask and protection key are as stated. However, there can be exceptions if system programs have changed the system mask or protection key.
2. You activate the TPF Assembler Debugger for VisualAge Client or TPF C Debugger for VisualAge Client code from this user exit by calling the UXMAC macro. Program load event detection code (the IUX_VADB UXMAC call) must be the **last** exit routine to be called in the exit. The debugger will **not** return to this user exit when it determines that the debugger must receive control. If you do not want the debugger to trace the program that will be entered, set the CE3_NO_DEBUG indicator in byte CE3DBCR (page 3 of the ECB) before the debugger is called. The debugger always sets this indicator off before activating the program.

3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Postinterrupt Processing

User exit routine UCCGPI is called from the user exit general postinterrupt routine (UXGPIR) if the exit point named GPI is active. UXGPIR allows a user-suspended ECB to be restarted when it is removed from a dispatching list. Exit point GPI is in CSECT CCUEXT.

Note: The UXGPIR program should only be invoked by an ECB-controlled program.

General Conditions at Entry

The registers at entry to UCCGPI are:

R0–R8	Contents unknown. The registers have been restored from the ECB register save area.
R9	Address of the ECB.
R10	Address of the General Postinterrupt routine UXGPIR.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. Do not deactivate this exit point while there are ECBs on a task dispatcher list that were intended to be passed to this user exit routine or the user routine will not receive control.
2. See “User Exit Routines - Common Entry Conditions” on page 6.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. Field CE1PSW in the ECB must contain a valid PSW on return to the NSI in the user exit interface because the user exit General Postinterrupt routine executes a load PSW from CE1PSW to pass control.
2. See “User Exit Routines - Common Return Conditions” on page 7.

Get Block With ECB

User exit routine UCCGBE is invoked by the Get Block With ECB routine after a core block has been allocated, and before returning to the caller. This exit point, named GBE, can be activated dynamically. Exit point GBE is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCGBE are:

R1	FCT (Frame Control Table) entry pointer.
R2	EVM address of the storage block.
R3	Address of routine invoking CL\$GET02.
R4	Pointer to address space flag.

Note: Equates have been provided to test the address space flag. See the user exit prologue in CUSR for more detailed information.

R9	Address of the associated ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM.
3. The ECB address, in R9, will be consistent with the address space.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Get Block without ECB

User exit routine UCCGBK is invoked by the Get Block Without ECB routine after a core block has been allocated, and before returning to the caller. This exit point, named GBK, can be activated dynamically. Exit point GBK is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCGBK are:

R1	FCT (Frame Control Table) entry pointer.
R2	SVM address of the storage block.
R3	Address of routine invoking CL\$GET01.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Get Common Block

User exit routine UCCGCB is invoked by the Get Common Block routine after a common block has been allocated and before returning to the caller. This exit point, named GCB, can be activated dynamically. Exit point GCB is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCGCB are:

R1	CCT (Common Block Control Table) entry pointer.
R2	SVM address of the common block.
R3	Address of routine invoking CL\$GCOMC.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM. Because the block address is common to both address spaces, an address space flag is not supplied or needed.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Get System Work Block

User exit routine UCCGSB is invoked by the Get System Work Block routine after a system work block has been allocated, and before returning to the caller. This exit point, named GSB, can be activated dynamically. Exit point GSB is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCGSB are:

R1	SCT (SWB Control Table) entry pointer.
R2	SVM address of the system work block.
R3	Address of routine invoking CL\$GSWBC.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM. Since the block address is common to both address spaces, an address space flag is not supplied or needed.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Library Function Call (ISO-C)

User exit routine UCCCLE is used with ISO-C support. It is activated on a call to an ISO-C library function if the exit point named CLE is active. User exit CLE is called from an ISO-C environment. Exit point CLE is in the library startup code (CSTRTL).

Activation of this exit point is performed selectively on library load modules. The UXCMC macro specifies the library load modules to be activated. For example, the user exit point could be activated for the ISO-C library but the activation must be done again for a user-written library. The status of library user exits is propagated across all versions of a library.

General Conditions at Entry

The registers at entry to UCCCLE are:

R1	Address of the C parameter list.
R2	The called function entry point address.
R3	The address of the called function name. If the function name is not available, R3 will be 0.
R4	Contains the length, in bytes, of the function name. If R3=0, R4=0.
R8	Points to the dynamic load module (DLM) header.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working Storage
Address state	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Library Function Return (ISO-C)

User exit routine UCCCLX is used with ISO-C support and is activated on return from an ISO-C library function if the exit point named CLX is active. User exit CLX is called from an ISO-C environment. Exit point CLX is located in the library startup code (CSTRTL).

Activation of this exit point is performed selectively on library load modules. The UXCMC macro specifies the library load modules to be activated. For example, the user exit point could be activated for the ISO-C library but not for a user-written library. The status of library user exits is propagated across all versions of a library.

General Conditions at Entry

The registers at entry to UCCCLX are:

R1	The value returned by the called function in R15 (if any).
R2	The called function entry point address.
R3	The address of the called function name. If the function name is not available, R3 will be 0.
R4	Contains the length, in bytes, of the function name. If R3=0, R4=0.
R8	Points to the dynamic load module (DLM) header.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working Storage
Address state	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

LODIC Macro

User exit routine UCCLODC is called whenever a LODIC macro call is processed. Exit point LODC is located near the very end of the LODIC macro processing routine contained in copy segment CICS of CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCLODC are:

R0	Return code passed from the user exit.
-1	Indicates that the ECB will be placed on the bottom of the suspend list. Control will return to the CPU loop.
0	Indicates that the system resources available are below the shutdown levels defined for the specified priority class (more work will not be started). Control will return to the application program.
1	Indicates that the system resources available are above the shutdown levels defined for the specified priority class (more work is allowed to be started). Control will return to the application program.
R1	Address of the SWB to add to the list if R0 is -1.
R7	Address of the macro parameters.
R8	Address of the program being run.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	EVM.

Programming Considerations at Entry

- See “User Exit Routines - Common Entry Conditions” on page 6.
- See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.
- The macro expansion contains the register index of the register that was supplied with the USRPRM= keyword. The value of that register can be obtained from the register save area in the ECB.

Programming Considerations on Return

For all registers except R0 and R1, see “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

- R0 can be changed to take a different action than was indicated on entry to UCCLODC.

- If R0 is changed to -1, you must put the address of the SWB to add to the suspend list into R1.

Online Mini Dump

User exit routine UCCOMD is invoked from system error processing if the OMD exit point is active and **ZASER DATAX** has been entered. UCCOMD allows you to check the ISO-C stack before dumping system data and to collect additional system data and add it to the system data that system error processing normally collects for the dump data user exit (CPSU).

Note: Do not use UCCOMD if you do not have the Online Mini Dump Facility.

General Conditions at Entry

The registers at entry to UCCOMD are:

R10 Pointer to the CPSE workarea.

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

Note: See a listing of CCCPSE for the contents of the remaining registers.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. User processing must not change the system state. Return must be made to the next sequential instruction.
2. See the UCCOMD prologue in CUSR for the sample code for adding additional data.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Pool Address Retrieval

User exit routine UCCGPFA is called during GETFC macro processing after the file address compute (FACE) program retrieves the pool address for the record you requested (if the GPFA exit point is active). Exit point GPFA is located in copy segment GRFS of CSECT CCSONP.

General Conditions at Entry

The registers at entry to UCCGPFA are:

R2	Indicates whether the pool address has been retrieved in a commit scope.
0	Indicates that the pool address was not retrieved in a commit scope; 0 indicates a normal retrieval.
1	Indicates that a commit scope is active.
R3	Pointer to the macro parameter list.
R7	Pointer to an 8-byte area that contains the pool file address.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem.
System mask	Unmasked.
Protect key	1.
Address space	EVM.

Programming Considerations at Entry

Pool file addresses that are retrieved in a commit scope are given back to the TPF system if the commit scope is rolled back.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Program Event Recording (PER)

User exit routine UCCPER is invoked whenever a PER program interrupt is taken that matches the address range and event type requested for display by the ZSPER command and the exit point named PER is active. Exit point PER is contained in copy segment CPER of CSECT CCCPSE.

The PER exit point allows you to interrupt data and expand the amount of data captured by the PER interrupt. The PER user exit logic issues a SNAPC (with return) to write the PER data to tape. When the PER user exit is not active, PER processing issues the SNAPC for the common set of data.

General Conditions at Entry

The registers at entry to UCCPER are:

R1	Address of a 1-byte return flag used to show whether the PER interrupt data should be processed or discarded by the ZSPER command.
R3	Address of the PER control area. The IDSPER macro can be used to DSECT the control block.
R6	Address of the storage block containing the PER interrupt data. The IDSPER macro can be used to DSECT the data block.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. A PER interrupt can occur in either the SVM or the EVM.
2. R9 may not contain a valid ECB address in the SVM.
3. The PER hardware interface locations are unchanged.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. To bypass additional processing of PER interrupt data by the ZSPER command, set bit 0 of the byte at the address in R1 to 1 (X'80'). To continue processing the PER interrupt data, no change is required. Regardless of the flag setting returned by this exit, the PER interrupt may still be eligible for processing by the PER debugging tools exit point (UCCPER2) or the TPF Assembler Debugger for VisualAge Client.
2. See “User Exit Routines - Common Return Conditions” on page 7.

Program Event Recording (PER) Debugging Tools

User exit routine UCCPER2 is invoked whenever a PER event matches the address ranges and event types specified in the IDSPER control area (for systemwide tools) or the entry control block (ECB) PER work area (for ECB-related tools) and the exit point named PER2 is active. Exit point PER2 is contained in copy segment CPER of CSECT CCCPSE.

The PER debugging tools exit point allows you to process the PER interrupt with one or more user-developed debugging tools. These tools will see the interrupt after ZSPER command support (if active), but before the TPF Assembler Debugger for VisualAge Client.

General Conditions at Entry

The registers at entry to UCCPER2 are:

R1	Previous pointer on the PER interrupt handler stack.
R3	Address of the PER control area. The IDSPER macro can be used to map the structure of the control block.
R6	Address of the storage area containing the PER interrupt data. The IDSPER macro can be used to map the structure of the data block.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. A PER interrupt can occur in either the SVM or the EVM.
2. Register 9 (R9) may not contain a valid ECB address in the SVM.
3. The PER hardware interface locations are unchanged.
4. If the user tools processing this PER interrupt are non-interactive display or recording tasks, return should be made to CPER so the TPF Assembler Debugger for VisualAge Client (if it is active) can also handle the PER interrupt.
5. If the user tools processing this PER interrupt are interactive, return will not be made to CPER and the interrupt will not be visible to the TPF Assembler Debugger for VisualAge Client. For this condition, correct functioning of the TPF Assembler Debugger for VisualAge Client may not be possible.
6. If it does not return control to CPER, the user exit must clean up the stack frame that was being used by the PER interrupt handler when the exit was invoked. (On entry, R1 contained the previous stack pointer to be stored into PFXPSAVE to complete stack cleanup.)
7. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

RCS I/O Queue Thresholding

User exit routine UCCTHR is provided as a default for processing a record cache subsystem (RCS) I/O queue threshold exceeded condition. This exit point, which is nondynamic, is invoked by entry point CJIVTTMR in control program CSECT CCRCSK whenever the I/O queue threshold value computed for an RCS subsystem is exceeded.

This user exit provides I/O queue depth statistics for a degraded record cache subsystem (RCS) controller on a periodic basis whenever a calculated threshold value is exceeded for the subsystem. System processing action depends on the action specified by the user exit in the thresholding parameter list provided.

An interface parameter list (IDSTHR) is provided by CJIVTTMR that passes relevant queue depth information. CJIVTTMR accepts the processing action that the user specifies in the parameter list on return.

General Conditions at Entry

The registers at entry to UCCTHR are:

R6 IDSTHR parameter list.

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Unmasked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. The parameter list IDSTHR is provided so the control program is the only code that is required to understand the internal system data structures. Because a menu of processing options is provided, you do not have to be aware of the internal data structures and, therefore, should not need to change them because of system changes.
2. Return must be made to CJIVTTMR because that entry point restarts the periodic (1 second) timer interval for threshold monitoring.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

1. R14 will contain a return code, set by the user exit routine, that will show what action is to be taken. The following return code definitions are supported:
 - 0 - No action required
 - 4 - Prime w/longest queue is down
 - 8 - Mod w/longest queue is down
 - 12 - Turn thresholding off
 - 16 - Change threshold value
 - 20 - Dupe w/longest queue is down

2. See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Release Block With ECB

User exit routine UCCRBE is invoked by the Release Block With ECB routine before releasing the core block. This exit point, named RBE, can be activated dynamically. Exit point RBE is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCRBE are:

R1	FCT (Frame Control Table) entry pointer.
R2	Address of the storage block.
R3	Address of routine invoking CL\$REL02.
R4	Pointer to address space flag.

Note: Equates have been provided to test the address space flag. See the user exit prologue in CUSR for more detailed information.

R9	Address of the associated ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM.
3. The ECB address, in R9, will be consistent with the address space.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Release Block without ECB

User exit routine UCCRBK is invoked by the Release Block Without ECB routine before releasing the core block. This exit point, named RBK, can be activated dynamically. Exit point RBK is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCRBK are:

R1	FCT (Frame Control Table) entry pointer.
R2	SVM address of the storage block.
R3	Address of routine invoking CL\$REL01.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Release Common Block

User exit routine UCCRCB is invoked by the Release Common Block routine before releasing the common block. This exit point, named RCB, can be activated dynamically. Exit point RCB is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCRCB are:

R1	CCT (Common Block Control Table) entry pointer.
R2	SVM Address of the common block.
R3	Address of routine invoking CL\$RCOMC.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM. Since the block address is common to both address spaces, an address space flag is not supplied or needed.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Release System Work Block

User exit routine UCCRSB is invoked by the Release System Work Block routine before releasing the system work block. This exit point, named RSB, can be activated dynamically. Exit point RSB is in copy segment CLHV of CSECT CCSTOR.

General Conditions at Entry

The registers at entry to UCCRSB are:

R1	SCT (SWB Control Table) entry pointer.
R2	SVM Address of the system work block.
R3	Address of routine invoking CL\$RSWBC.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. No core block management data records can be modified.
2. The exit point can be invoked in either the SVM or the EVM. Since the block address is common to both address spaces, an address space flag is not supplied or needed.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

RIAT

The Record ID Attribute Table (RIAT) contains information on every file type (both fixed and pool) in the database. For more information on this table, see “Record ID Attribute Table” on page 373.

User exit routine UCCRIT is invoked from the RITID macro call if the user exit option was specified for the ID during RIAT definition. This exit point, named RIT, **cannot** be activated dynamically. Exit point RIT is in copy segment CEFJ of CSECT CCFADC and copy segment CVF3 of CSECT CCVFAC.

The RIAT exit allows you to intercept all FIND/FILE macros after a RIAT entry has been found. This is a nondynamic exit point, but the exit may or may not be activated depending on the RIAT indicators for that ID.

General Conditions at Entry

The registers at entry to UCCRIT are:

R1	Address of the MIOB set up to service the request.
R2	Address of the RIAT entry, if found. If the MIOB was setup from the RIAT default entry R2 will be set to zero (0).
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

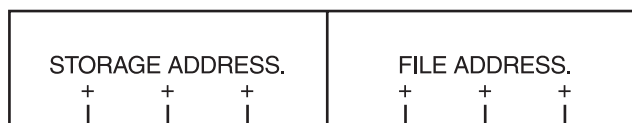
System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address space	EVM.

Programming Considerations at Entry

1. Filing critical user records.

As many as six records per subsystem can be designated for filing during catastrophic error processing. CTIN generates a list of six doublewords for each subsystem that holds record addresses. These addresses are copied to disk as part of CPSF.

The CINFC tag CMMZFA is used to access the list. The leftmost fullword contains the 31-bit main storage address of the record to be filed and the rightmost fullword contains the record file address in FARF format, as shown in the accompanying diagram.



If fewer than six records are filed for any subsystem, X'FF' must appear in byte 4 of the doubleword following the last record entry for that subsystem.

2. The contents of program registers R0–R7 have been saved in the ECB register save area.
3. See “User Exit Routines - Common Entry Conditions” on page 6.

4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.
5. See *TPF System Generation* for RIAT START call restrictions before you code this user exit and observe the following:
 - When the RIAT bits are changed as part of this user exit, do not set the bits such that they conflict with the rules for the RIATA macro as stated in *TPF System Generation*.
 - Do **not** set or reset the SIMMED and SDELAY values in this user exit because the setting or resetting of these bits must be seen by all processors at the same time or database corruption may occur.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

ROUTC

User exit routine UCCRTC is invoked from the ROUTC and SENDC macros.

This exit point, named RTC, **cannot** be activated dynamically. Exit point RTC is located in copy segment CLXA of CSECT CCCCCP1. Copy segment CT20 of CSECT CCCTIN detects if the user exit routine exists. If so, CT20 turns on the active indicator for this exit point.

UCCRTC allows you to intercept outbound messages and modify the message text and the routing parameters.

General Conditions at Entry

The registers at entry to UCCRTC are:

R1	Address of message block.
R2	Address of RCPL.
R3, R4	Set to 0.
R7	Address of ROUTC parameter list.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address space	EVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

For all registers except registers R3 and R4, see “User Exit Routines - Common Return Conditions” on page 7.

Registers R3 and R4 on return from UCCRTC are:

R3=0	Continue processing without process selection vector (PSV) activation.
R3=-1	Stop processing the message. Release the message from the data level, if still holding the block, and return to the calling program. If the exit code takes control of the message block by using the \$DISBC macro, it is responsible for clearing the data level held indicator.
R3=RID (resource ID) with high order bit on	Activate the PSV routine using the RID.

R3=address Activate the PSV routine using the PSV name at the specified (EVA) address.

If R3 shows that a PSV routine is to be activated:

R4=address The 3 bytes of data at the specified address (EVA) are placed in the ECB fields EBCM02-4 of the new ECB used to invoke the PSV routine.

R4=0 No data is passed in ECB fields EBCM02-4 of the new ECB used to invoke the PSV routine.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

SNAPC Error

User exit routine UCCSPX is called from SNAPC error processing at the end of processing just before returning to the calling program. UCCSPX will not be called when the return to CPU loop option has been set at the UCCSNP user exit. Exit point SPX is in CSECT CCCPSE.

General Conditions at Entry

The registers at entry to UCCSPX are:

R1	Address of a 1-byte return flag that is used to show whether control should return to the CPU loop or skip the SNAP dump processing.
R2	For a SNAPC (with return), the address of a doubleword field that contains the PSW for the NSI. For a SNAPC (with exit), the address of the SNAP9000 routine in CPSM.
R3	Pointer to an address that contains the program old PSW at the time of the SNAPC, followed by 16 fullwords that contain the registers at the time of the SNAPC.
R4	The SVA address of the ECB, or 0 if the SNAPC took place without an ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

Refer to a listing of CCCPSE for the contents of the remaining registers.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. Do not issue the TIMEC and CWRTC macros or others which issue the \$MONTC macro.
2. Do not issue macros which can give up control.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

SNAPC Error Entry

User exit routine UCCSNP is called at the beginning of the SNAPC error processing routine if the exit point named SNP is active. Exit point SNP is located in CSECT CCCPSE.

General Conditions at Entry

The registers at entry to UCCSNP are:

R0, R4–R9	Unknown
R1	Address of a 1-byte return flag used to show whether control should return to CPU loop or skip the SNAP dump processing.
R2	Pointer to SNAPC parameters.
R3	Pointer to an address that contains the program old PSW at the time of the SNAPC, followed by 16 fullwords that contain the registers at the time of the SNAPC.
R10	Base register for CCCPSE
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

Note: The contents of all the registers at the time the error occurred have been saved by CCCPSE. Refer to a listing of CCCPSE for the exact location.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM or EVM.

Programming Considerations at Entry

1. The program old PSW and the program interrupt code can be found in their assigned fixed storage locations.
2. Do not issue the TIMEC and CWRTC macros or others which issue the \$MONTC macro.
3. Do not issue macros which can give up control.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Split Access

If the exit point named FSC is active then user exit routine UCCFSC is invoked during FACE or FACS processing after the split for the record being requested has been located. Exit point FSC is located in copy segment CUSR of CSECT CCUEXT.

General Conditions at Entry

The registers at entry to UCCFSC are:

R0	Input ordinal number.
R4	Pointer to split for requested record.
R6	Pointer to split chain header for requested record type.
R7	Output area.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	1
Address space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

Programming Considerations on Return

The following registers can be modified on return:

R0	Input ordinal number
R6	Pointer to split chain header for requested record type

Note: R13 must contain a return code set by the user exit routine to indicate the action to be taken. The following return codes are supported:

- | | |
|---|--|
| 0 | Continue normal processing. |
| 4 | The contents of R0, R6, or both have been modified; retry the FACE or FACS call. |
| 8 | Return an error indication to the FACE or FACS call. |

See “User Exit Routines - Common Return Conditions” on page 7 for additional return conditions.

General Conditions on Return

User processing must return to NSI. For more programming considerations, see “User Exit Routines - Common Return Conditions” on page 7.

Split Chain Header Access

If the exit point named FHD is active then user exit routine UCCFHD is invoked during FACE or FACS processing after the split chain header for the record type being requested has been located. Exit point FHD is located in copy segment CUSR of CSECT CCUEXT.

General Conditions at Entry

The registers at entry to UCCFHD are:

R0	Input ordinal number.
R4	Pointer to FACE table header.
R6	Pointer to split chain header for requested record type.
R7	Output area.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	1
Address space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

Programming Considerations on Return

The following registers can be modified on return:

R0	Input ordinal number.
R6	Pointer to split chain header for requested record type.

Note: R13 must contain a return code set by the user exit routine to indicate the action to be taken. The following return codes are supported:

- 0 Continue normal processing.
- 4 The contents of R0, R6, or both have been modified; retry the FACE or FACS call.
- 8 Return an error indication to the FACE or FACS call.

See “User Exit Routines - Common Return Conditions” on page 7 for additional return conditions.

General Conditions on Return

User processing must return to the NSI. For more programming considerations, see “User Exit Routines - Common Return Conditions” on page 7.

Stack Overflow Processing (ISO-C)

User exit routine UCCCSOX is activated at the end of stack overflow processing if the exit point named CSOX is active. User exit CSOX is called from an ISO-C environment. Exit point CSOX is in CSECT CCISOC (CIS2).

General Conditions at Entry

The registers at entry to UCCCSOX are:

R8	Points to the dynamic load module (DLM) header.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage
Address state	EVM.

Programming Considerations at Entry

1. Usually, the system state, system mask, and protection key are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state, changed the system mask, or changed the protection key.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Stack Overflow Processing Entry (ISO-C)

User exit routine UCCCSOE is activated on entry to stack overflow processing if the exit point named CSOE is active. User exit CSOE is called from an ISO-C environment. Exit point CSOE is in CSECT CCISOC (CIS2).

General Conditions at Entry

The registers at entry to UCCCSOE are:

R8	Points to the dynamic load module (DLM) header.
R9	Address of the ECB.
R13–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage
Address state	EVM.

Programming Considerations at Entry

1. Usually, the system state, system mask, and protection key are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state, changed the system mask, or changed the protection key.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Suspend ECB

User exit routine UCCSUSE is called whenever an ECB that is marked as a suspendable ECB (by means of the LODIC or TMSLC macros) will be suspended during CPU loop dispatch time. Exit point SUSE is located in copy segment CLHL of CSECT CCCLHR.

General Conditions at Entry

The registers at entry to UCCSUSE are:

R1	Address of the SWB that will be placed on the suspend list.
R9	SVM address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O and external interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

- See “User Exit Routines - Common Entry Conditions” on page 6.
- See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

None.

Suspend List Post-Interrupt

User exit routine UCCSUSP is called whenever an item is taken off the suspend list and is about to be dispatched. Exit point SUSP is located in copy segment CLHL of CSECT CCCLHR.

General Conditions at Entry

The registers at entry to UCCSUSP are:

R1	Address of the SWB that is taken off the suspend list.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O and external interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

- See “User Exit Routines - Common Entry Conditions” on page 6.
- See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.
- The application registers are not yet restored from the SWB.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

The application registers are restored and the ECB is given control.

Suspend List Resource Checking

User exit routine UCCSUSC is called indirectly by two different routines:

- The post-interrupt routine, which determines if an item will be added to the suspend list (see copy member CCIT of the CCNUCL CSECT)
- The suspend list post-interrupt routine, which determines if an item will be dispatched from the suspend list (see copy member CLHL of the CCCLHR CSECT).

Exit point SUSC is located in copy segment CICS of CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCSUSC are:

R0	Return code that is passed back to the post-interrupt routine or the suspend list post-interrupt routine. 0 There are not enough resources to create more work. <ul style="list-style-type: none">• If this is a new item, place it on the suspend list.• If the item is already on the suspend list, keep it there. 1 There are enough resources to create more work. <ul style="list-style-type: none">• If this is a new item, do not place it on the suspend list.• If the item is already on the suspend list, dispatch it from the suspend list.
R2	Index register for LODIC shutdown table (CPLODTAB).
R7	Address of parameter list: Byte 0 = X'41' Byte 1 = LODIC priority class indicator Byte 3 = Block type ignore flags.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O interrupts
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

- See “User Exit Routines - Common Entry Conditions” on page 6.
- See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.
- You can use the user exit with the LODIC macro user exit (UCCLODC).

Programming Considerations on Return

For all registers except R0 and R1, see “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

R0 can be changed to take a different action than was indicated on entry to UCCSUSC.

SVC Macro (Immediate)

User exit routine UCCSVX is invoked by SVC macro processing routines that return to the issuer of the macro by an LPSW from the SVC old PSW location (for example, no wait or implied wait). Control is passed to UCCSVX just before issuing the LPSW instruction (if the exit point named SVX is active). Exit point SVX is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCSVX are:

R0–R7	The contents have been restored from the ECB and contain the data that is being returned to the E-type program that issued the macro.
R8	Base of the E-type program that issued the macro.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	EVM.

Programming Considerations at Entry

1. To preserve the integrity of the stack, user processing must **not** issue an SVC macro.
2. The adjusted SVC old PSW, which points to the return address, can be found in PFXSVPWS.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

SVC Macro (Wait or Implied Wait: Postinterrupt)

User exit routine UCCSVW is invoked by those SVC macros that have a wait or implied wait, and that return to the issuer of the macro by an LPSW CE1PSW instruction in a postinterrupt routine. UCCSVW is invoked just before issuing the LPSW instruction if the SVW exit point is active. Exit point SVW is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCSVW are:

R0–R7	The contents have been restored from the ECB and contain the data that is being returned to the E-type program that issued the macro.
R8	Base of the E-type program that is to receive control.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	SVM or EVM.

Programming Considerations at Entry

1. Usually, the address space active at the time of the user exit depends on the processing of the postinterrupt routine. Most postinterrupt routines can complete all of their processing in the SVM. For some interrupt routines it is necessary to switch to the EVM, where they will complete their processing. They are not required to switch back to the SVM before returning to the macro decoder.
2. To preserve the integrity of the stack, user processing must **not** issue an SVC macro.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

SVC Macro Decoder

User exit routine UCCSVC is invoked from the SVC Macro Decoder for all SVCs, both direct and indexed, if the exit point named SVC is active. Exit point SVC is in CSECT CCMCDC.

General Conditions at Entry

The registers at entry to UCCSVC are:

R6	Address of the macro entry in the macro decoder table.
R8	Base address of E-type program that issued the SVC macro.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O Interrupts
Protect key	0
Address state	EVM.

Programming Considerations at Entry

1. The contents of program registers R0–R7 have been saved in the ECB register save area.
2. To preserve the integrity of the stack, user processing must **not** issue an SVC macro because this would cause the stack register to be reinitialized and the current saved data to be lost.
3. The SVC exit point occurs before the formatting of the entry in the macro trace table.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

System Error

User exit routine UCCSRX is called from system error processing after dump processing has completed and the system is ready to return to the NSI, CPU loop, or to exit the ECB.

Note: UCCSRX is not called if the *return to CPU loop* flag was set in the UCCSER user exit.

Exit point SRX is in CSECT CCCPSE. UCCSRX is provided to control the location to which return is made when system error processing has completed.

The user exit will be called once on the failing I-stream.

General Conditions at Entry

The registers at entry to UCCSRX are:

R6 Base address of the I-stream status save area for the executing I-stream. See the DCTISV DSECT. The failing program's resume PSW and registers may be extracted from DCTISV.

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

Refer to a listing of CCCPSE for the contents of the remaining registers.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. If you do not return to the next sequential instruction, you must enable the system for interrupts.
2. To pass control to the original destination set by CPSE in the resume PSW, return from the user exit routine.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

System Error Entry

User exit routine UCCSER is invoked at the beginning of the system error processing routine if the exit point named SER is active. Exit point SER is located in CSECT CCCPSE.

General Conditions at Entry

The registers at entry to UCCSER are:

R0, R2–R5, R9

Unknown

R1 Address of a 1-byte return flag used to show whether control should return to NSI or the CPU loop following error processing.

R6 Pointer to the DCTISV save area for the current I-stream.

R7–R8, R10 Base registers for CCCPSE.

R11–R15 See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address Space	SVM.

Programming Considerations at Entry

1. The program old PSW and the program interrupt code can be found using the DCTISV savearea. R6 contains the address of this area and the DCTISV DSECT maps it out.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

1. You must not attempt a direct return to the CPU loop in this exit. An interface is provided to request that control pass to the CPU Loop, instead of NSI, after completion of System Error processing. For more information, see the “Programming Considerations” comments for this exit in CSECT CCUEXT.
2. See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

TMSLC Macro

User exit routine UCCTMSL is called whenever a TMSLC macro call is processed. Exit point TMSL is located near the very end of the TMSLC macro processing routine contained in copy segment CICS of CSECT CCNUCL. This exit point is called before doing one of the following:

- Issuing a SNAPC dump with exit
- Suspending the ECB
- Returning to the application program.

General Conditions at Entry

The registers at entry to UCCTMSL are:

R0	Code passed to the user exit: <ul style="list-style-type: none">0 Return to the application program with no error.-1 Unrecognized time-slice name. Enter the SNAPC dump if the NOTFND= label is not coded.-2 Too many time-slice ECBs active. Enter the SNAPC dump if the EXCD= label is not coded.-3 Suspend the ECB, or if the macro indicates that the DISABLE parameter was used, force the ECB to lose control.
R1	Address of the SWB to add to the suspend list if R0 is -3.
R7	Address of the macro parameters.
R8	Address of the program that is being run.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	0
Address Space	EVM.

Programming Considerations at Entry

- R0 reflects the action that the system will take on return from UCCTMSL.
- See “User Exit Routines - Common Entry Conditions” on page 6.
- See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

For all registers except R0 and R1, see “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

- R0 can be changed to take a different action than was indicated on entry to UCCTMSL.

- If R0 is changed to -3, you must put the address of the SWB to add to the suspend list into R1.

TPFAR

Direct exits for TPFAR SQL calls are not provided. The method for trapping SQL calls is indirect and can be done with the ENTRC exit, UCCENTR. Trap all ENTRCs to program CRDA. CRDA handles all SQL calls and is only entered by application programs issuing SQL requests.

TPFDF Macro Trace Call

User exit routine UCCDFFC is invoked by copy segment UFZ0 of CSECT CCNUCL before the invocation of each TPFDF fast-link case. This exit point, DFFC, can be activated dynamically and is in copy segment CUSR of CSECT CCUEXT.

General Conditions at Entry

The registers at entry to UCCDFFC are:

R0–R7	Contents as set by the program executing the fast-link call.
R8	Program index (bits 0–15) and case index (bits 16–31).
R9	Address of the ECB.
R10	Unknown.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working storage
Address Space	EVM.

Programming Considerations at Entry

1. Bits 0–15 of R8 contain the fast-link segment index. See data macro IFLDDF for a mapping of the index number to the segment name.
2. Bits 16–31 of R8 contain the fast-link case number, offset by decimal 16. To obtain the actual case number, subtract 16.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

TPFDF Macro Trace Return

User exit routine UCCDFFR is invoked by copy segment UFZ0 of CSECT CCNUCL on return from each TPFDF fast-link case. This exit point, DFFR, can be activated dynamically, and is in copy segment CUSR of CSECT CCUEXT.

General Conditions at Entry

The registers at entry to UCCDFFR are:

R0–R8	Contents as set by the fast-link case on exit.
R9	Address of the ECB.
R10	Unknown.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem
System mask	Unmasked
Protect key	Working storage
Address Space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Trace C User Data

User exit routine UCCCTRC is invoked by the C function trace *program entry breakpoints* and *program exit breakpoints* service routine (CTR2) and by the C function trace other breakpoints service routine (CTRY). This user exit is invoked after the trace entry is inserted into the C function trace table, to allow the user to insert additional trace data into the optional C function trace user area.

General Conditions at Entry

The registers at entry to UCCCTRC are:

R2	TCA address
R3	For <i>program entry breakpoints</i> - DSA address of the caller For <i>program exit breakpoints</i> - current DSA address For other breakpoints - current DSA address
R5	CID address
R6	Breakpoint type, defined in IDSCTR
R8	Current DLM address
R9	ECB pointer
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Unknown
System mask	Unmasked
Protect key	Unknown
Address space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

- You can issue the ENATC macro from the CTRC user exit. However, the ENATC macro uses register 15 to provide return code information. This use of register 15 conflicts with the user exit usage of register 15 for addressing. Therefore, before you issue the ENATC macro from the CTRC user exit, establish addressability using a base register other than 15.

Note: You cannot issue the SETTC macro while processing the CTRC user exit routine because the trace table environment has already been set up.

- If the offline dump processing code references the user data words (which are the output in registers 5 and 6) as storage locations, you should use storage offsets instead of storage addresses. The reason for this restriction is that the offline dump processing loads the C function trace area from tape into MVS storage, which is not the original TPF storage addresses.

Programming Considerations on Return

User data first full word optionally updated by the user exit; the value in R5 will be saved in the first word of the trace entry field (ICTRE_USR) upon return from the user exit.

User data second full word optionally updated by the user exit; the value in R6 will be saved in the second word of the trace entry field (ICTRE_USR) upon return from the user exit.

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Trace Environment Customization

User exit routine UCCCDEB is invoked by the C function trace exception routine (CTR0). The ISO-C user can use UCCCDEB to customize the C function trace environment.

General Conditions at Entry

The registers at entry to UCCCDEB are:

R2	TCA address
R3	Current DSA address
R5	CID address
R8	Current DLM address
R9	ECB pointer
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor state
System mask	Masked for I/O interrupts
Protect key	0
Address space	EVM.

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

You can issue the SETTC macro or the ENATC macro from the CDEB user exit. However, the SETTC and ENATC macros use register 15 to provide return code information. This use of register 15 conflicts with the user exit usage of register 15 for addressing. Therefore, before you issue the SETTC macro or the ENATC macro from the CDEB user exit, establish addressability using a base register other than 15.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Transaction Log Write

User exit routine UCCWLOG is called at the point where writing a record to the log buffer has been completed (if the WLOG exit point is active). Exit point WLOG is in copy segment CL20 of CSECT CCTLOG.

General Conditions at Entry

The registers at entry to UCCWLOG are:

R1	Pointer to the WLOGC CONTROL structure.
R2	Pointer to the WLOGC DATALIST structure.
R7	Pointer to the WLOGC parameter list.
R8	Base of the E-type program that issued the request.
R9	Address of the entry control block (ECB).
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked for I/O
Protect key	Zero (0)
Address space	EVM or SVM.

Programming Considerations at Entry

1. The entry address space, SVM or EVM, depends on the address space of the caller of WLOGC. WLOGC can be called from either real-time or CP code.
2. See “User Exit Routines - Common Entry Conditions” on page 6.
3. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Update Tape Display

User exit routine UCCLDD issues a Load Display command to update the LED display on a tape device. UCCLDD is invoked from the exit point LDD, which is nondynamic, in one of the following routines:

- Routine LOCATEYYY of segment CEDT in CSECT CCCPSE.
- Routine CCDSW100 of segment CEDT in CSECT CCCPSE.

General Conditions at Entry

The registers at entry to UCCLDD are:

R0	R0 must have one of the following values:
X'00'	Equated to LDDTPSW, indicates the exit was called during system error processing and came from routine CCDTPSW1 in segment CEDT.
X'04'	Equated to LDDYYY, indicates the exit was called during system error processing and came from routine LOCATEYYY in segment CEDT.
Note: This should be taken into consideration when adding code to the exit.	
R4	Pointer to the TSTB section 2 entry for the tape device in question
R6	Pointer to the TSTB section 1 entry for the tape device in question
R7	Pointer to the Load Display workarea located in the calling segment
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. This routine is invoked during system error processing
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. If the user exit returns condition code zero, either Message 1, Message 2, or both messages will be displayed.
2. If the exit returns a non-zero condition code, no message will be displayed.

User Header Label

User exit routine UCCUHL is invoked from the BUILDHDR routine if the exit point named UHL is active. UCCUHL allows you to modify tape header labels. Exit point UHL, which is nondynamic, is located in copy segment CEFM of CSECT CCTAPE.

User exit routine UCCUHL contains a control routine, UHLSTART, that calls user label subroutines that you can modify.

The user header label control routine calls subroutines UHL1, UHL2, ... UHL8 in order, until all 8 subroutines are called, or until any one of the subroutines fails to set up the user label properly. A properly setup user header label will start with the characters UHLx where x is the current label number. After verifying the label, it is written.

Note: If you use the user header label routines user exit, you should also use the real-time user label routines user exit in segment UXTH. See “User Label Routines” on page 233 for more information.

General Conditions at Entry

The following registers are set up for the user by each of the user label subroutines:

R2	Pointer to the IBM standard HDR1 and HDR2 labels.
R3	Pointer to User Label Work Area (mapped by ITUHL).
R4	Pointer to the TLMR (Tape Label Mask Record) entry.
R5	Pointer to the TSTB/1 (Tape Status Table/Section 1) entry.
R6	Pointer to the TSTB/2 (Tape Status Table/Section 2) entry.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

When UXTH is called to write user header records, the TSTB/3 entry for the tape is not yet set up. Therefore, to be consistent with UXTH, R4 has a pointer to the TLMR record. This requires you to use the equates in TAPEQ to access any fields in the TLMR.

When the first user label subroutine gets control (UHL1, UTL1, or IHL1), registers R2, R4, R5, and R6 will be set up as listed previously. R3 will point to the base of the user label work area (ULWA). It is recommended that you do not change this register. However, if it is necessary to use R3, the pointer to the ULWA can be found in CE1CR3. R7 will contain the return address and this value must not be changed. R0 and R1 can be used by the user label subroutine.

When any other user label subroutines get control, register R3 will once again point to the base of the previous user label work area. R0, R1, R2, R4, R5, and R6 will all contain whatever values they contained on exit from the previous user label subroutine.

Data Structures

DSECT ITUHL maps a 1055-byte working area that is used by UXTH. The working area is broken up into the following areas:

- ULWALAB1 through ULWALAB8 correspond to the 8 user label definitions. These areas are to be filled in by the UHL and UTL routines, and will contain user labels read from input tapes for the IHL routines.
- ULWAUSR is a user work area. This area is provided for temporary storage because ECB fields are not to be modified by the user label subroutines.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

User Trace Area Initialization

User exit routine UCCCEXP is invoked by the C function trace exception routine (CTR0). The C user can use UCCCEXP to initialize storage after the user trace area storage is allocated. The SETTC macro must have been previously issued by this ECB to allocate the C function trace user area (for example, in the CDEB user exit).

General Conditions at Entry

The registers at entry to UCCCEXP are:

R2	TCA address
R3	Current DSA address
R5	CID address
R8	Current DLM address
R9	ECB pointer
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor state
System mask	Masked for I/O interrupts
Protect key	0
Address space	EVM

Programming Considerations at Entry

See “User Exit Routines - Common Programming Considerations” on page 7.

You can issue the ENATC macro from the CEXP user exit. However, the ENATC macro uses register 15 to provide return code information. This use of register 15 conflicts with the user exit usage of register 15 for addressing. Therefore, before you issue the ENATC macro from the CEXP user exit, establish addressability using a base register other than 15.

Note: You cannot issue the SETTC macro while processing the CEXP user exit routine because the trace table environment has already been set up.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

Validate Tape for Output

User exit routine UCCVTO validates that a tape can be used for output before it is automatically used by automount routines as an ALT tape. UCCVTO is invoked from the exit point VTO, which is nondynamic, in one of the following routines:

- Routine VOL1INT of segment CEFA in CSECT CCTAPE.
- Routine CHECKDATE of segment CEDT in CSECT CCCPSE.

General Conditions at Entry

The registers at entry to UCCVTO are:

R0	R0 must have one of the following values: X'00' Equated to VTOCEDT, indicates the exit was called during system error processing and came from segment CEDT. Note: This should be taken into consideration when adding code to the exit. X'04' Equated to VTOCEFA, indicates the exit came from segment CEFA.
R1	Pointer to the IBM standard VOL1 label (addressable by the equates in TAPEQ).
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. See “User Exit Routines - Common Entry Conditions” on page 6.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. If the user exit returns condition code zero, the process of mounting the tape as an ALT tape continues.
2. If the exit returns a nonzero condition code, the tape is not mounted as an ALT tape and a message informs the operator that the tape is not valid for output. If the user exit is activated from CEFA, the COSK0288E message is sent. If the user exit is activated from CEDT, the CPSE0027E message is sent.
3. See “User Exit Routines - Common Return Conditions” on page 7.

WAITC Macro Entry

User exit routine UCCWAI is called at the beginning of macro processing for the WAITC macro (if the exit point named WAI is active). Exit point WAI is in CSECT CCNUCL.

General Conditions at Entry

The registers at entry to UCCWAI are:

R0–R7	Contents as set by the program that issued the WAITC macro.
R8	Base of E-type program that issued the WAITC.
R9	Address of the ECB.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Problem (see “Programming Considerations at Entry”, item 1)
System mask	Unmasked (see “Programming Considerations at Entry”, item 1)
Protect key	Working storage or zero (0)
Address space	EVM.

Programming Considerations at Entry

1. Usually, the system state and system mask are as stated. However, there can be exceptions if system programs issued MONTC to get to supervisor state and changed the system mask.
2. The WAI exit point occurs before checking the I/O count in the ECB. Therefore, the I/O count is unknown at entry to user processing.
3. See “User Exit Routines - Common Entry Conditions” on page 6.
4. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

1. If the I/O counter is zero (no I/O is pending) when UCCWAI returns to the NSI in WAITC macro processing, no other user exit routines are invoked and control is returned to the E-type program that issued the WAITC. If the I/O counter is not zero (I/O is pending) when UCCWAI returns to the NSI, the MONWC macro causes the user exit routine at label UCCSVW to run. When all I/O is complete for the ECB, UCCSVW is run at post interrupt time (if the SVW exit point is active) and control is returned to the E-type program that issued the WAITC.
2. See “User Exit Routines - Common Return Conditions” on page 7.

WTOPC Message Translation

User exit routine UCCWTOP is invoked from control program (CP) copy member CEDI to allow you to translate individual characters in output message text. CEDI normally translates the plus sign (+) and greater than sign (>) into periods (.). UCCWTOP allows you to translate the plus sign (+) and greater than sign (>) into characters other than periods (.) before CEDI translates them.

General Conditions at Entry

The registers at entry to UCCWTOP are:

R5	The number of bytes, from 1 to 256, to be translated.
R7	Pointer to the first byte of text to be translated.
R8	Pointer to the primary terminal address for this message.
R11–R15	See “User Exit Routines - Common Entry Conditions” on page 6.

System Conditions at Entry

System state	Supervisor
System mask	Masked
Protect key	0
Address space	SVM.

Programming Considerations at Entry

1. This routine can be called several times for a single message depending on the length of the message. Therefore, the message header may not be part of the message text to be translated.
2. See “User Exit Routines - Common Programming Considerations” on page 7 for other considerations.

Programming Considerations on Return

See “User Exit Routines - Common Return Conditions” on page 7.

General Conditions on Return

See “User Exit Routines - Common Return Conditions” on page 7.

ECB-Controlled Program User Exits Overview

User exits allow you to add user-unique processing at various points in TPF programs without having to modify the released programs. ECB-controlled program user exits are called by ECB-controlled program exit points that reside in ECB-controlled programs.

Note: Because ECB-controlled program exit points that are provided are in function programs, no performance impact on the system is anticipated. However, have detailed knowledge of TPF system internals before activating or writing user exit code.

Exit Points

Exit points are predefined locations in TPF system processing from which user-unique processing code can be invoked. This user-unique code (called *user exit*, *user exit routine*, and *user processing*) will then execute as an extension of a TPF system function.

Exit point status is not carried across an IPL. The ZSTIM command can be used during restart, cycle-up, or cycle-down to automatically activate or deactivate exit points.

A dynamic E-type program exit point is one that can be set on or off by SYSTC switches. A nondynamic E-type program exit point does not change. It is always on (that is, it is always called).

Along with the exit points, sample programs or stubs have been provided. As released by IBM, most of these stubs simply issue a BACKC, while some provide sections of sample code. You must change the program stub to an operational program to perform whatever processing is required.

Selective Activate Exits

Selective activate exits allow you to limit the use of an E-type loader loadset to a specific ECB origin. An ECB origin can be a terminal address, communication line number, port number, user ID, NCP ALS, NCP name, and others.

To use the selective activate function you must:

- Activate the selective activation function during system generation.
- Create an enable command.
- Create a disable command.

Activating the Selective Activation Function

You can activate the selective activation function in the CONFIG macro or by using the ZSYSG command. When activated, selected ECBs can enter the programs in a selectively activated loadset. Otherwise, no ECBs can enter the programs in a selectively activated loadset. See *TPF Operations* for more information on ZSYSG.

Creating an Enable Command

To enable loadsets for specific ECB origins, create a command that builds two core-resident structures; a selective activation table and a selective activation index. The selective activation table should contain loadset names and selective activation

numbers. The selective activation index should contain the ECB origins and pointers to the loadset names. Entries should be added to these structures using a user-defined enable message. To do this your command should do the following:

1. Add the ECB origin to the selective activation index.
2. If the loadset name is already in the selective activation table, update the index.
3. If the loadset name is not in the selective activation table, add the entry and call the selective activate utility (COLW) to set the activation number. Update the index.

Note: COLW returns an activation number of zero if the loadset has not been selectively activated.

4. Update the file resident structure (if one exists) to allow enable requests to survive an IPL.

Figure 2 shows an example of a selective activation table. Figure 3 shows an example of a selective activation index.

Loadset Name	Selective Activation Number
-----	-----
Joseph	0
Sally	4
Fred1	8

Figure 2. Selective Activation Table Example

Note: In Figure 2, Joseph was enabled to be used from an ECB origin, but is not yet activated selectively. Loadsets Sally and Fred1 were selectively activated and were assigned activation numbers 4 and 8, respectively.

Terminal Addr	Index into Selective Activation Table
-----	-----
020103	0
030567	0, 1
002203	1
292834	1, 2

Figure 3. Selective Activation Index Example

Note: Figure 3 contains a list of ECB origins (terminal addresses) and the associated index into the selective activation table. In this example, ECBs originating on terminal 292834 can enter programs from loadsets Sally or Fred1.

Creating a Disable Command

The user-defined disable command should stop an ECB from using a loadset. This message should remove the index pointer for the selective activation index entry, and if there are no more index entries that reference the specified loadset, remove the associated selective activation table entry (if there are no more ECB origins that reference a loadset, there is no longer a need to maintain the activation number in the selective activation table). If there are no more loadsets enabled to be used by the specified ECB origin, then the index entry can be removed. By removing the table index from the index entry, ECBs originating from that origin will not have the specified loadset's activation number in their activation number list (which is used

by Enter / Back to determine which version of a program to use). If a file copy of the mapping structure is maintained, then it will have to be updated to reflect the disable request.

Note: It is entirely possible that the loadset name will not be found in either table. The user-written disable code should handle this condition. This can occur for 2 reasons:

1. The loadset was selectively activated, but never enabled. Here, a table entry for the loadset was never created.
2. It is possible that the loadset **was** enabled; however, the mapping structure is not recorded on file (for example, the **enables** do not survive an IPL). If an IPL were to occur after the enable and before the disable request, the disable code would not find the loadset in the tables.

Data Macros to Develop User Exits

The following data macros have been provided to assist in the development of the user exits in ECB-controlled programs:

BK0UX	An extension of the recoup keypoint, BK0RP. It is also called by BK0RP. It can be used to subdivide the recoup user area, BK0USR.
CZ1UX	An extension of the system error equates, CZ1SE. It is also called by CZ1SE. It can be used to define user system error equates for the error numbers DB0000 through DBFFFF. These numbers have been set aside for user definition.
UXTEQ	Called by the BEGIN macro for all ECB-controlled user programs including the user exit programs. It can be used to define user-specified system equates.

E-type User Exit Considerations

When ECB-controlled program user exits are called using the ENTRC macro, you must decide whether to return to the calling program, save and restore registers, and/or maintain data areas. Some functions might not complete if control is not returned to the calling segment with the registers intact.

User Exit Allocation and Activation

Table 1 refers to the activation and allocation of user exits for ECB-controlled programs. Note which user exits are allocated through SIP before you load your online system.

Note: Copy members are activated differently than E-type programs. Copy members are not activated because copy members are copied to parent code which is activated.

Table 1. Activation and Allocation of User Exits for ECB-Controlled Programs

System Function	User Exit	Description	Allocation	Activated by
APPN	UACP	E-type program	Allocated	ENTRC
	UALS	E-type program	Allocated	ENTRC
	UAPN	E-type program	Allocated	ENTRC
	UARG	E-type program	Allocated	ENTRC

Table 1. Activation and Allocation of User Exits for ECB-Controlled Programs (continued)

System Function	User Exit	Description	Allocation	Activated by
C/C++ Language Dynamic Link Library (DLL) Support	USUD	E-type program	Allocated	DLL load in CISO (Note 7)
Clocks	GCALX	Copy member	None	Initiating program
	GDATX	Copy member	None	Initiating program
	GCLKX	Copy member	None	Initiating program
Communications Source Common Exit	UCS1	E-type program	Allocated	ENTRC (Note 1)
Continuous Data Collection	CDCA	E-type program	Allocated	ENTRC
	CDCB	E-type program	Allocated	ENTRC
Database reorganization	UBDB	E-type program	Allocated	ENTRC
Deadlock Detection	CLUD	E-type program	Allocated	CL40
Dump Data User Exit	CPSU	E-type program	Allocated	ENTNC
Dynamic LU	CDLY	E-type program	Allocated	ENTRC
ECB display	UDE0	E-type program	Allocated	ENTRC
E-type loader display	UELG	E-type program	Allocated	ENTRC
Extra Program Record Report	UELI	E-type program	Allocated	ENTRC
Get global environment lists	UENV	E-type program	Allocated	ENTRC
HPR support	URTP	E-type program	Allocated	ENTRC
Log Recovery Error Processing	CL99	E-type program	Allocated	ENTRC
LU 6.2	CSXA	E-type program	Allocated	CREMC/ENTNC (Note 5)
	CSXB	E-type program	Allocated	CREMC/ENTNC (Note 5)
	CSXC	E-type program	Allocated	CREMC/ENTNC (Note 5)
Message Queue Interface Channel Exits	(Note 6)	E-type program	Allocated	ENTRC
Message Router	COBC	E-type program	Allocated	ENTRC (Note 3)
Module copy select/validate	UCPY	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX0	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX1	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX2	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX3	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX6	E-type program	Allocated	ENTRC
Pool migration and conversion	UPX7	E-type program	Allocated	ENTRC
Record loadset history	UELL	E-type program	Allocated	ENTRC
Record program history	UELM	E-type program	Allocated	ENTRC
Recoup	BRU1	E-type program	Allocated	ENTxC
	BRU2	E-type program	Allocated	ENTxC
	BRU3	E-type program	Allocated	ENTxC
Restrict ZOLDR input	UELD	E-type program	Allocated	ENTRC
Selective activate	UELF	E-type program	Allocated	ENTRC
Selective activate	UELH	E-type program	Allocated	ENTRC (Note 4)
Selective activate	UELN	E-type program	Allocated	ENTRC

Table 1. Activation and Allocation of User Exits for ECB-Controlled Programs (continued)

System Function	User Exit	Description	Allocation	Activated by
Selective activate	UELX	E-type program	Allocated	ENTRC (Note 4)
Selective activate	UELW	E-type program	Allocated	ENTRC
Selective activate	UELX	E-type program	Allocated	ENTRC (Note 4)
Selective activate	UEL1	E-type program	Allocated	ENTRC (Note 4)
Selective core resident load	UCLB	E-type program	Allocated	ENTRC
SLC Communication Source	CIM2	E-type program	Allocated	ENTxC
SNA Communication Route Selection	CSJV	E-type program	Allocated	ENTxC
SNA Message Recovery	CMVU	E-type program	Allocated	ENTxC
SNMP	UCOM	E-type program	Allocated	ENTRC
	UMIB	E-type program	Allocated	ENTRC
System Error Message Control	CPSD	E-type program	Allocated	ENTxC
System Message Processing	UOP1	E-type program	Allocated	ENTxC
	UOP2	E-type program	Allocated	ENTxC
System Restart	URS1	E-type program	Allocated	ENTxC
	URS2	E-type program	Allocated	ENTxC
System State Change	USC1	E-type program	Allocated	ENTxC
	USC2	E-type program	Allocated	ENTxC
	USC3	E-type program	Allocated	ENTRC
	USC4	E-type program	Allocated	ENTRC
Tape Display Setup	UXTD	E-type program	Allocated	ENTxC
Tape Library Validation	CORU	E-type program	Allocated	ENTxC
TPF Files System Initialization	UBOT	E-type program	Allocated (Note 2)	Initiating program (CBOT)
TPF MQSeries	CUIR	E-type program	Allocated	CREEC
TPF MQSeries	CUIT	E-type program	Allocated (Note 7)	Initiating program (MQ MCA)
TPF MQSeries	CUIV	E-type program	Allocated (Note 7)	Initiating program (MQ QMGR)
TPF MQSeries	CUIW	E-type program	Allocated (Note 7)	Initiating program (MQ QMGR)
TPF MQSeries	CUIA	E-type program	Allocated (Note 7)	Initiating program (MQ QMGR)
TCP/IP Support	CLA4	E-type program	Allocated	ENTNC
	CLCH	E-type program	Allocated	CREEC
	CLCI	E-type program	Allocated	CREEC
	CLCV	E-type program	Allocated	CREDC
	CLCX	E-type program	Allocated	CREMC
	CLCM	E-type program	Allocated	ENTRC
	CLCQ	E-type program	Allocated	ENTRC
	CLCS	E-type program	Allocated	ENTRC
	CLCU	E-type program	Allocated	ENTRC

Table 1. Activation and Allocation of User Exits for ECB-Controlled Programs (continued)

System Function	User Exit	Description	Allocation	Activated by
	C542	E-type program	Allocated	ENTRC
	UACC	E-type program	Allocated	ENTRC
	UMATAL	E-type program	Allocated (Note 7)	ENTRC
	UMATAS	E-type program	Allocated (Note 7)	ENTRC
	UMATCH	E-type program	Allocated (Note 7)	ENTRC
	UMATFI	E-type program	Allocated (Note 7)	ENTRC
	UMATRO	E-type program	Allocated (Note 7)	ENTNC
	UMATSE	E-type program	Allocated (Note 7)	ENTRC
	UMATSS	E-type program	Allocated (Note 7)	ENTRC
	UMATTR	E-type program	Allocated (Note 7)	ENTRC
	USOK	E-type program	Allocated	ENTRC
User Command Processor	UME1	E-type program	Allocated	ENTNC
User Command Table	UMET	E-type program	Allocated	Read only
User Data Recovery Copy Support	UDRS	E-type program	Allocated	ENTDC
User Data Recovery Restore Support	UDRR	E-type program	Allocated	ENTDC
User input device support	UELC	E-type program	Allocated	ENTRC
User Label Routines	UXTH	E-type program	Allocated	ENTxC
User library function	UELE	E-type program	Allocated	ENTRC
VFA restart	CVFX	E-type program	Allocated	ENTRC
VIPA processor deactivation	UVIP	E-type program	Allocated	ENTRC
Virtual reader support	UELB	E-type program	Allocated	ENTRC
VisualAge for TPF Debuggers	CDBPUX	E-type program	Allocated	ENTRC
VisualAge for TPF Debuggers	CDBUXT	E-type program	Allocated	ENTRC
WTOPC page control	UOP3	E-type program	Allocated	ENTRC
3270 Welcome Screen	CSLJ	E-type program	Allocated	ENTRC

Notes:

1. You must set the associated SYSTC switch on to invoke the user exit.
2. Allocated in the initiating program.
3. You must set the SYSTC switch SBCOMXT on to invoke the user exit. SBCOMXT can be set on by coding COMEXIT=YES on the MSGRT macro in SIP.
4. You must set the associated SYSTC switch on to use the selective activate function. See "Activating the Selective Activation Function" on page 109 for more information.

5. This user exit is entered with a CREMC or an ENTNC depending on which TPF program calls it.
6. The name of the E-type program activated by the MQI channel exits is specified by the user with the ZMQID DEFINE command. See *TPF Operations* for more information.
7. Allocated as part of a dynamic link library (DLL) or dynamic link module (DLM).

ECB-Controlled Program User Exits

This chapter describes each TPF entry control block (ECB) user exit, how to use the user exit, and the following information concerning each user exit:

- Input
- Programming considerations
- Return values.

Check OLDR Load Deck

The check OLDR load deck exit, UELR, performs additional offline checking against the program names and versions specified in the OLDR load deck. UELR is called by COLR in the OLDR offline program.

Input

- The number of parameters that were specified in the **PARM=** parameter. This is taken from the load deck EXEC statement.
- A pointer to an array of character strings, each string represents a parameter found after the **PARM=** parameter. This is taken from the load deck EXEC statement.
- The name of the program and its 2-character version code taken from the input load deck card.

Programming Considerations

UELRL is shipped in skeleton form.

Return Values

- If return=0, no messages are issued by the caller.
- If return=*odd* (for example, 1, 3, or 5), the following error message will be issued by the caller of this routine and the program will not be loaded:

```
xxxx0099E: COLR_VALIDATE_PROGRAM_CARD:
            USER_EXIT ERROR yyyy ENCOUNTERED FOR
            PROGRAM zzzzzz
            <additional text, such as "loadset ignored">
```

- If return=*nonzero, even* (for example, 2, 4, or 6), the following warning message will be issued by the caller of this routine and the program will not be loaded:

```
xxxx0099W: COLR_VALIDATE_PROGRAM_CARD:
            USER_EXIT ERROR yyyy ENCOUNTERED FOR
            PROGRAM zzzzzz - PROGRAM LOADED ANYWAY
```

Notes:

1. xxxx is the job name (for example, OLDR)
2. yyyy is the return code from this routine
3. zzzzzz is the program name and version code.

Clock Global Update Exits

The clock exits are used to update the subsystem user global clock and calendar fields. They are copy members that get expanded in other ECB-controlled clock segments. Each contains sample code which copies the global fields from the first subsystem user to the rest of the subsystem users in the subsystem. If the system does not contain multiple subsystem users, the sample code will return control to the expanding (or parent) segment.

Exit	Parent Segment	Function
GCALX	CDTC	Initializes the global calendar fields when the system is cycled above 1052 state.
GDATEX	CDTD	Updates the global calendar fields whenever a midnight boundary is crossed.
GCLKX	GLBL	Updates the global clock fields every time a minute boundary is crossed.

The interface requirements for each user exit are described in the prologue of its parent segment.

Command Manager

The command manager user exit, UELD, determines if an ECB has the authority to issue a specified ZOLDR command.

UELD, is called by ZOLDR commands.

Input

Input message block on D0.

Programming Considerations

- UELD is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.

Return Values

Void.

Common Symbol Table

The common symbol table user exit, UCST, allows you to add data macros (DSECTs), or any symbols that are considered common to real-time assembler programs, to the common symbol table to be referenced when debugging all programs. Use of the common symbol table eliminates the need for multiple copies of the DSECT and symbol information for each real-time assembler program being retained and loaded to the TPF system. Instead, only one copy of the common DSECTs or symbols are kept in the TPF system.

See the ucst.asm program for an example of the common symbol table.

Input

All symbols defined in UCST are included in the common symbol table and are used as an exclusion list to build an application program's local symbol table.

Programming Considerations

- The common symbol table is referenced during the online symbol lookup process and is used in the offline symbol table generation process. If a symbol is defined in the common symbol table, it will be eliminated from the local symbol table for the program.
- If a symbol is relocatable and found in the common symbol table, the symbol lookup process resolves the relocation by finding the base register in the local symbol table for the program. For example, if you added DSECT MYTBL to the common symbol table and TBLFLD1 is a data field in MYTBL, the symbol lookup process uses the displacement of TBLFLD1 from the common symbol table and uses the base register that is defined in the current program to resolve symbol TBLFLD1.
- You can have multiple versions of the common symbol table in the TPF system by loading multiple versions of the table through the online loader. The TPF Assembler Debugger for VisualAge Client uses the application ECB activation number as the basis to select the common symbol table with the same or lower activation number.
- Be careful when determining which DSECTs or symbols you want to add to the common symbol table and maintain the DSECT or symbol in the common symbol table once you have added it. Removing entries from the common symbol table will result in lost symbol definitions. For example, DSECT MYTBL is in the common symbol table and all data fields in DSECT MYTBL are excluded from the local symbol table for the program. If you remove DSECT MYTBL from the common symbol table, the TPF Assembler Debugger for VisualAge Client cannot resolve any data fields in MYTBL from the local symbol table or the common symbol table until the local symbol table is rebuilt to include data fields for MYTBL.
- You **must** run the TPFSYM offline program against UCST to create the SYSADATA file. This file is used as an exclusion list for offline symbol table creation (TPFSYM) and as a common symbol table during symbol lookup.

Return Values

None. However, system error 0ADB12 is issued and the ECB exits whenever an attempt is made to enter UCST. See *Messages (System Error and Offline)* for more information about system errors.

Communications Source Common

This exit routine, UCS1, is activated by communications source programs before normal processing but not if entered after processing by other communications source programs (that is, only one exit per input message). This exit can be used to examine the input message origin.

Input

R0	Contains CIAA to show activation from CIAA
R1	Pointer to input message block.
R4	Address of the WGTA entry for the message origin
D0	Message block in communication message format
D1	The AAA record, if in use
D3	The RCB record, if in use
EBW042	Set to X'08' if the input was from a 3270, otherwise set to X'00'.

Input

R0	Contains CITT to show activation from CITT
R1	Pointer to input message block
R2	Pointer to RVT1 entry
R3	Pointer to RVT2 entry
D0	Message block in communication message format
EBW000	Input RCPL.

Programming Considerations

- UCS1 is a dynamic exit and is only called when SYSTC switch SBUCS1 is set on.
- UCS1 is shipped as a BACKC stub.
- Communications Source programs (CIAA and CITT) enter UCS1 with return expected.
- Communications Source program CINN, for local 3270s, completes its processing in CIAA and, therefore, activates UCS1 through CIAA.
- The input block, from CIAA, will include an input message (assembled) or answerback.

Continuous Data Collection Information Storage

The continuous data collection information storage user exit, CDCB, allows you to store user data into tables that were previously defined by using the continuous data collection table creation user exit, CDCA.

CDCB is called during the processing of the ZDCDO command with the START parameter specified. CDCB runs immediately after system data collection is completed.

Input

The variables that are passed in CDCB are:

- | | |
|-----------------|---|
| CDC_KEY | A pointer to a 9-character string that corresponds to the 8-character representation of the CPU ID plus the null character. |
| CDC_TIME | A pointer to a 27-character string that corresponds to a structured query language (SQL) time stamp. |

Programming Considerations

- You must store CDC_KEY and CDC_TIME into each record to relate the data in that record with the data collected by the TPF system.
- You can store any data that is collected in the tables defined with CDCA.
- CDCB must be returned to the caller in the same state that it was on entry.

Return Values

None.

Continuous Data Collection Table Creation

The continuous data collection table creation user exit, CDCA, allows you to create tables that are necessary to store data.

CDCA is called during the processing of the ZCDCO command with the CREATE parameter specified.

Input

None.

Programming Considerations

- Any tables that you create must provide columns for CDC_KEY and CDC_TIME. These pointers serve as common keys for the remaining data and allow you to relate a row in one table to a row in another table.
 - CDC_KEY is a pointer to a 9-character string that corresponds to the 8-character representation of the CPU ID plus the null character.
 - CDC_TIME is a pointer to a 27-character string that corresponds to a structured query language (SQL) time stamp.
- CDCA must be returned to the caller in the same state that it was on entry.

Return Values

None.

CP-CP Session Activation

The CP-CP session activation user exit, UACP, determines if a TPF processor is allowed to start CP-CP sessions. When no TPF processor in the loosely coupled complex has CP-CP sessions, UACP is called during cycle-up and when an APPN-capable link is activated while the TPF system is in NORM state.

Note: If this user exit is coded to not allow CP-CP sessions on a processor, you can still start CP-CP sessions on that processor by issuing the ZNETW ACT command.

Input

CE1CPD	CPU ID of this processor (the processor that wants to start CP-CP sessions)
---------------	---

Programming Considerations

- IBM provides sample code for UACP, which sets the return code to 0 indicating that this processor is allowed to start CP-CP sessions.
- In a uniprocessor TPF environment, you do not need to modify this user exit because the default logic shipped by IBM allows CP-CP sessions on the TPF processor that invokes this user exit.
- If you want to limit the processors that are allowed to start the CP-CP sessions in a TPF loosely coupled complex, code UACP to check the CPU ID to determine if this TPF processor is allowed to start CP-CP sessions.
- Data level D2 must not be modified. All other data levels are available for use by this user exit.
- EBW000–EBW103 and CE1UR1–CE1UR7 must not be modified. EBX000–EBX103 is available for use by this user exit.

Return Values

R0 contains the return code:

- | | |
|---|--|
| 0 | This TPF processor is allowed to start CP-CP sessions. |
| 1 | This TPF processor is not allowed to start CP-CP sessions. |

Database Reorganization

The database reorganization (DBR) user exit (UBDB) is activated by the DBR initialization program BDBF after BDBF completes the initialization of the DBR keypoints for one subsystem user. The DBR user exit enables users to complete any further initialization of the DBR keypoints for a particular subsystem user.

Input

D0	DCTBPK parameter list
D1	Output message block for BDBN
D2	DBR master keypoint
D3	IDSFCZ parameter block (used by ZDBSO INIT and ZDBRO INIT triplet messages)
D4	Record types not to be captured by DBR
D5	DBR exception record (for the last record type in the system)
D6	ESFAC core block (used by ZDBSO INIT and ZDBRO INIT triplet messages)
EBW000–EBW015	Subsystem user IDs for each subsystem user entered with the ZDBSO INIT SSU1/SSU2... message
EBW060–EBW063	Number of subsystem user keypoints initialized with the ZDBSO INIT SSU1/SSU2... message
EBW064–EBW067	Number of subsystem users entered with the ZDBSO INIT SSU1/SSU2... message
EBW074	Type of IPL (X'04' = IPL from general file)
EBW088–EBW089	Number of processors generated in the system
EBW092	Processor ordinal number entered with the ZDBRO INIT triplet message
EBW096	Hexadecimal value for the I-stream entered with the ZDBRO INIT triplet message
EBX000–EBX009	Input message (first two tokens)
EBX060–EBX063	Number of subsystem users to be processed by the ZDBSO INIT message
EBX064–EBX067	Pointer to the current subsystem user table entry (used by ZDBSO INIT message)
EBXSW0	FINWC switch – X'80'
EBXSW2	First subsystem user's DBR keypoints initialized – X'04' (used by the ZDBSO INIT message)
EBXSW3	ZDBSO switch – X'80'

EBXSW4	BYPASS=YES option entered – X'80'
EBXSW4	ZDBSO INIT SSU1/SSU2... message entered – X'40'
EBXSW4	ZDBRO INIT triplet message entered – X'20'
EBXSW5	First triplet combination for the ZDBSO INIT message processed – X'FF'

Programming Considerations

- Data levels 3, 4, 5, and 6 can be used, but CRUSA macros must be issued for each of the data levels that is to be used by UBDB.
- Data levels 0, 1, and 2 must **not** be used.
- ECB work area fields and switches specified previously must **not** be used by UBDB.

References

TPF Database Reference.

Deactivate Phase I Selective Activate

The deactivate phase I selective activate exit, UELW, makes it possible to check if there are references to the loadset. UELW is called from segment COLA.

UELW is only called when the FORCE parameter is not issued with the ZOLDR DEACTIVATE command.

Input

Loadset name (blank padded on right).

Programming Considerations

- UELW is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.
- See “Selective Activate Exits” on page 109.

Return Values

- If return=FALSE, deactivate processing stops and exits.
- If return=TRUE, deactivate processing continues.

Deactivate Phase II Selective Activate

The deactivate phase II exit, UELF, updates a user-defined structure when a selectively activated loadset is deactivated.

UELF is called by the ZOLDR DEACTIVATE SEL command.

Input

- Loadset name
- Activation number.

Programming Considerations

- UELF is shipped in skeleton form.
- All data levels must be returned to the caller in the same state they were upon entry.
- See “Selective Activate Exits” on page 109.

Return Values

Void.

Deadlock Detection

The deadlock detection user exit, CLUD, is called by segment CL40 with a C function call to perform deadlock notification.

Input

The registers at entry to CLUD are:

R6 Address of both the entry control block (ECB) and the input/output block (IOB) involved in the deadlock condition.

Programming Considerations

This exit provides sample code for a return code of 8.

Return Values

R15 contains the return code:

- 0** Processing continues as if the exit was never called.
- 4** Processing ends with an error. The service routine for the ZECBL command (with the E parameter specified) is called to remove all the IOBs associated with this ECB and to schedule a D9 dump.
- 8** The CE1SUD and CE1SUG fields of each ECB that is involved in the deadlock condition are set to indicate that a deadlock has occurred. The post-interrupt routine in the IOB is also activated.

DEARRANGE_CTK9 (UPX1)

The DEARRANGE_CTK9 user exit, UPX1, performs any needed accounting or utility functions when keypoint 9 (CTK9) is filed by the TPF system through the CYYA interface. You can also provide your own DEARRANGE_CTK9 function to convert CTK9 from the current format in processor storage to pool expansion (PXP) format or to a user-defined format that is compatible with PXP format.

The DEARRANGE_CTK9 user exit is called from the DEARRANGE_CTK9 function in segment CYH1.

Input

EBXSW0	Contains the index of the data level containing CTK9 in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format. The index is in the form of data level 0 (D0) to data level E (DE). EBXSW0 is only valid when EBXSW2=X'80'.
EBXSW1	Contains switches used by CYYA. These switches can only be queried by this user exit.
EBXSW2	If EBXSW2=X'00', CTK9 is on data level F (DF) in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format. If EBXSW2=X'80', CTK9 is on the data level specified by EBXSW0.
Data Level F	If EBXSW2=X'00', data level F (DF) contains CTK9 in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format. If EBXSW2=X'80', data level F (DF) is free.

Return Values

R6=0	Default return code. If this value is returned, CYH1 converts CTK9 from its 32-way loosely coupled pool support format to PXP format. Use this return code if you code this user exit but do not change CTK9.
R6=1	If this value is returned, CYH1 does not change CTK9. CYH1 immediately returns to its caller. Use this return code if you code this user exit and convert CTK9 from its current format in processor storage to PXP format or to a user-defined format that is compatible to PXP format.

Programming Considerations

- This user exit is called by CYH1 when the conversion mode indicator (CY1MD32) in CTK9 is PXP or CONVERTING. If the conversion mode indicator is 32LC or FALLING_BACK, CYH1 does not call this user exit.
- As shipped by IBM, this user exit issues a BACKC macro to return to the caller. CTK9 remains unchanged.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller.
- If R6=0 on return to CYH1, CTK9 must be in 32-way loosely coupled pool support format and it must be on the same data level where it was on entry to the user exit.

- If R6=1 on return to CYH1, CTK9 must be on data level F (DF) and it must be in PXP format or a user-defined format that is compatible with PXP format.
- If EBXSW2=X'80' on entry to the user exit, CTK9 must remain unchanged on the data level specified in EBXSW0 on return from the user exit. If R6=1, a second CTK9 in pool expansion (PXP) format or a user-defined format that is compatible to PXP format must reside on data level F (DF).

References

See *TPF Database Reference* for more information about pool support.

Debug Registration

The debug registration user exit, CDBPUX, is part of the TPF Assembler Debugger for VisualAge Client or TPF C Debugger for VisualAge Client and allows you to verify user registration information before the entry is stored in the debug tables.

CDBPUX is called from segment CDBP, which is the entry point function for CDBP.

Input

IPAddress The address of the workstation that submitted the registration request.

setup_message A pointer to the registration information. The structure definition for the registration information is contained in the `c$cdbg.h` header file.

Programming Considerations

1. IBM ships this user exit as a C function coded with a return of TRUE.
2. CDBPUX is an object file that is linked into CDBP, which is a C dynamic load module (DLM). The interface to CDBPUX is a C function call; the user exit code can be written in assembler if the TMSPC and TMSEC macros are used for the corresponding prolog and epilog.

Return Values

TRUE The workstation and the registration information are correct. On return from CDBPUX, CDBP stores the information in the appropriate trace table.

FALSE The workstation or the registration information is not correct. On return from CDBPUX, CDBP ignores the registration information and sends an error message to the workstation that submitted the request.

Detect Selective Activate Support

The detect selective activate support exit, UEL1, is shipped by IBM to return FALSE. If FALSE is returned by this user exit, you cannot activate loadsets selectively. If user exits are developed to support selective activation, then this user exit must be changed to return a value of TRUE. UEL1 is called when Activate Phase I detects a selective activation request.

If an installation has chosen not to use selective activation, then the operator will get the following message:

```
"CEL1 - ACTIVATE ENDED - LOADSET xxxxxxxx - SELECTIVE  
ACTIVATION NOT SUPPORTED"
```

Input

None.

Programming Considerations

- UEL1 is shipped in skeleton form, with one return statement.
- See "Selective Activate Exits" on page 109.

Return Values

- If return=FALSE, selective activate user exits have not been developed at this installation (IBM shipped return value).
- If return=TRUE, selective activate user exits have been developed at this installation.

Display

The display exit, UELG, allows you to display your own additional data. UELG is called by the ZOLDR DISPLAY commands.

Input

Address of uelg_zoldr_display_parms structure that contains the parameters available to this function.

Programming Considerations

UELG is shipped in skeleton form, with 1 return statement.

Return Values

Void.

DNS Select an IP Address

The Domain Name System (DNS) Select an Internet Protocol (IP) address user exit, UDNS, enables incoming TCP/IP connections to be load balanced across a TPF loosely coupled complex and provides two ways to answer a DNS query (address) request. The user exit is passed the TPF host name from the DNS request and a list of active IP addresses for the host name and their associated central processing unit (CPU) and IP type of information. You have the following three choices for selecting an IP address:

- Select an IP address from the list
- Select a CPU ID, where the TPF system will select an IP address from the list on the selected CPU
- Let the TPF system select an IP address from the list automatically.

Input

The following four input parameters are passed to the user exit to determine which IP address will be passed back to the client:

- A pointer to a structure defined in `netdb.h` containing a list of IP addresses and associated processor IDs. This structure contains:
 - The IP address.
 - The CPU ID.
 - The IP address type.
- A pointer to the host name from the DNS query.
- The number of active IP addresses associated with the input host name.
- A pointer to the output buffer. See Return Values for the format of this buffer.

Programming Considerations

The TPF server will select an IP address by a round-robin process if this user exit does not select one.

Return Values

A pointer to a structure defined in `netdb.h` containing one of the following:

- An IP address (associated with the input host name)
- A CPU ID (the TPF system chooses the IP address from the CPU ID)
- Zeros (the TPF system chooses the IP address from the list associated with the host name).

Dump Data

The Dump Data user exit, CPSU, is called by system error processing. This exit provides an interface for dumps with an associated entry control block (ECB) that will enable a user to save dump data for on-line processing. Both SERRC and SNAPC dumps are supported. Dump data will be presented to the dump data user exit. Activating the dump data user exit does not bypass normal system processing.

Input

- Dump data from system error dumps (EBXSW0 = X'00')

The data is chained in 4KB blocks in SNAPC format, parsed according to the data name, and attached to the following chains:

– CE1CR0

Data Name	Content
-----------	---------

DCTERI	First dump block: <ul style="list-style-type: none">- Error message- Registers- Program- Prog old PSW- LN/IA/TA- Subsystem
---------------	---

or MESSAGE	If the dump is a duplicate dump, appended messages are dumped. No other data is provided.
-------------------	---

– CE1CR1

Data Name	Content
-----------	---------

ECB	ECB all pages: (if any) <ul style="list-style-type: none">- ECB data fields- ECB MACRO TRACE
------------	---

DECBFRAM	Frame containing active and inactive data event control blocks (DECBs). Zero or more occurrences possible.
-----------------	--

– CE1CR2

Data Name	Content
-----------	---------

PROGRAM	Program block (if any)
----------------	------------------------

D0	Data level 0 (if any)
-----------	-----------------------

D1	Data level 1 (if any)
-----------	-----------------------

D2	Data level 2 (if any)
-----------	-----------------------

...	...
-----	-----

DF	Data level F (if any)
-----------	-----------------------

DLIBLOCK	Data block that is attached to the TPFDF SW00SR slot. Zero or more occurrences possible
-----------------	---

DECBBLK	Core block that is attached to a DECB. Zero or more occurrences possible.
----------------	---

– CE1CR3

Data Name	Content
-----------	---------

DETACO	Detached data block Level 0. Zero or more occurrences possible
---------------	--

- | | |
|-----------------|--|
| DETAC1 | Detached data block Level 1. Zero or more occurrences possible |
| ... | ... |
| DETACF | Detached data block Level F. Zero or more occurrences possible |
| DETACDEC | Detached data block from a DECB. Zero or more occurrences possible |
- CE1CR4
- | | |
|------------------|--|
| Data Name | Content |
| C/STACK | Initial C stack frame (if any) |
| C/STACK | Current C stack frame (if any) |
| C/STATIC | Current C static block (if any) |
| AUTOSTOR | Current auto storage block (if any) |
| SW00SR | Current TPFDF parameter block (if any) |
- Dump data from snap dumps (EBXSW0 = X'FF')
- The snap data blocks are chained to CE1CR0 and parsed using the IDSSNP DSECT.

Programming Considerations

1. The released version of CPSU contains sample code which will release the dump data blocks and exit the ECB.
2. Issuing a SNAPC dump in dump data processing will cause an infinite loop. Use the appropriate tests to terminate the loop when coding a SNAPC macro.
3. The dump data user exit cannot be controlled independently for SERRC and SNAPC.
4. A SYSTC tag, SBDTAC, is defined for this user exit. The tag specifies whether detached data blocks are presented. If the switch is on, there are detached data blocks. The maximum number of detached blocks, including both ECB data levels and DECBs, is specified by the DETDATAAX constant in CZOCP.
5. You can use the IDECB DSECT to map the DECB frames that are presented.

Dynamic LU

The dynamic logical unit (LU) user exit, CDLY, allows you to specify the values that are used to create resource definitions for new LU resources that log on to the TPF system.

You can also use this user exit to specify which LU resources, if any, can log on to the TPF system.

This user exit is called only if an LU resource tries to log on to the TPF system and a resource definition does not already exist for that LU resource. Therefore, the dynamic LU user exit can be called when:

- An LU resource logs on to the TPF system.
- An operator enters the ZNCNS INITIALIZE command for an LU 6.2 resource.
- The ROUTC macro is used to send a message to an application on another processor and the receiving processor creates a resource definition.
- Functional management message routing (FMMR) support is used to send a message to an application in another complex and the receiving processor creates a resource definition.

Input

- R1** Address of the parameter list, which contains the following:
- Four-byte address of a copy of the resource vector table (RVT) entry for the LU resource, which is defined by the RV1VT DSECT.
See Table 4 on page 140 for information about the fields in the copy of the RVT entry that you can change with this user exit.
 - Four-byte address of an area that contains the name of a process selection vector (PSV) routine for the LU resource. (The PSV name is blank when this user exit is called.)
See “Programming Considerations” for information about specifying the name of a PSV routine for the LU resource.

Programming Considerations

- D0–D7 cannot be modified.
- EBX096–EBX099 cannot be modified.
- To avoid session initiation logic errors, do **not** allow the entry control block (ECB) to give up control during processing of CDLY.
- Control is returned to the CDLX segment, which verifies the changes made by this user exit.
- Table 2 shows the OSTG options for each LU type (except X.25) that can be changed using this user exit. The OSTG options that can be changed are indicated by the word EXIT. The OSTG options that are not relevant or not supported are indicated by a dash (—). See *TPF ACF/SNA Network Generation* for a description of the OSTG options.

Table 2. LU Types and User-Specified OSTG Options

OSTG Option	LU Types												
	3277	3278	3284	3286	3287	3288	3289	3614	360X	APSLU	APPC	FMMR	NEF
LEID=	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT
UMODE=	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT
PSV=	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	—	—	—	EXIT

Table 2. LU Types and User-Specified OSTG Options (continued)

OSTG Option	LU Types												
	3277	3278	3284	3286	3287	3288	3289	3614	360X	APSLU	APPC	FMMR	NEF
AWARE=	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT	EXIT
CHAIN=	—	—	—	—	—	—	—	—	—	—	—	—	—

- Table 3 shows the OSTG options for each X.25-type resource that can be changed using this user exit. The OSTG options that can be changed are indicated by the word EXIT. REQUIRED indicates that you must specify a value for the OSTG option. OPTIONAL indicates that you can use the default value or specify a new value for the OSTG option. The OSTG options that are not relevant or not supported are indicated by a dash (—). See *TPF ACF/SNA Network Generation* for a description of the OSTG options.

Table 3. X.25 LU Types and the Permitted OSTG Options

OSTG Option	LU Types					
	MCHLU	VCLU	AX001	AX002	XALCI	FTPI
LEID=	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (REQUIRED)	EXIT (REQUIRED)	EXIT (OPTIONAL)	EXIT (OPTIONAL)
UMODE=	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)
PSV=	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (OPTIONAL)	EXIT (REQUIRED)	EXIT (OPTIONAL)
AWARE=	EXIT (OPTIONAL)	EXIT (OPTIONAL)	—	—	—	—
CHAIN=	—	EXIT (OPTIONAL)	—	—	—	—

- Table 4 describes the fields in the copy of the RVT entry that you can change with this user exit.

Table 4. Description of Fields in the Dynamic LU User Exit

Field	Description	Default Value
RV1LEIDF	<p>A 4-byte field that corresponds to the LEID= OSTG option. This field contains the 1-byte length of the logical end-point identifier (LEID) followed by the 1- to 3-byte LEID.</p> <p>If you specify an LEID for a 3270, AX001, or AX002 LU resource, use the following format, which will be verified when control is returned to the CDLX segment:</p> <p>3270 Specify a 3-byte LEID for 3270 devices that will log on to non-SNA applications (that is, applications that use an LEID to address a terminal).</p> <p>AX001 Specify a 2-byte LEID that consists of the line number and interchange address of the terminal.</p> <p>AX002 Specify a 1-byte LEID that consists of the line number of the terminal.</p> <p>Note: If you specify an LEID for any other LU type, the format is <i>not</i> verified when control is returned to the CDLX segment.</p>	X'0000'
RV1MODE3	A 1-byte field that corresponds to the UMODE= OSTG option.	X'00'

Table 4. Description of Fields in the Dynamic LU User Exit (continued)

Field	Description	Default Value
SNAWARE bit in RV1MODE1	A bit that corresponds to the AWARE= OSTG option.	Off
SNACHAIN bit in RV1MODE2	A bit that corresponds to the CHAIN= OSTG option.	Off
RV1DVTP2	<p>A 1-byte field that identifies the X.25 LU type, which can be one of the following:</p> <p>TP1MCH Multichannel link</p> <p>TP1VCL Virtual circuit</p> <p>TP1XA1 Airlines X.25 type 1</p> <p>TP1XA2 Airlines X.25 type 2</p> <p>TP1XALC Airlines line control interconnection (ALCI)</p> <p>TP1FTP Fast transaction processing interface (FTPI).</p> <p>Note: You must specify a value for this field when using X.25 LU resources.</p>	None

See *TPF ACF/SNA Network Generation* for a description of the OSTG options.

- Use the PSVNAME field in this user exit to specify the name of a PSV routine for the LU resource. The PSVNAME field is a 6-byte field that corresponds to the PSV= OSTG option.

Note: If you specify a PSV name for an LU resource, that PSV routine **must** be defined to the TPF system in at least 1 offline ACF/SNA table generation (OSTG) RSC statement. See *TPF ACF/SNA Network Generation* for more information about the OSTG program and the RSC statement.

- This user exit is shipped by IBM to return the value 2 in R0 for every LU resource that tries to log on to the TPF system using dynamic LU support. This means that no LU resources can log on to the TPF system using dynamic LU support until you update this user exit.

Return Values

R0 contains the return code:

- 0** Success
- 2** LU resource not authorized to log on to the TPF system.

ECB Display

The ECB display user exit, UDE0, allows you to exclude ECBs from the active ECB display. UDE0 is activated by the ZFECB command.

Input

R1 Address of the line of text (as defined by the IDFET macro) that contains information about the active ECB.

Programming Considerations

- UDE0 is shipped in skeleton form.
- If you want to exclude an ECB from the active ECB display, set EBXSW1 to X'80'.

Return Values

None.

Extra Program Record Report

The extra program record report exit, UELI, makes it possible to report on how many extra program records are available. Extra program records are obtained from the #XPRGn fixed file record type. This user exit also makes it possible to report on how many #APRGn records (used to store ADATA files) are available.

UELI is called by:

- The online portion of the general file loader (ACPL) at the very end of the load process
- The online portion of the auxiliary loader (CIL1) after the E-type programs have been loaded
- The E-type loader at the end of ZOLDR ACCEPT phase 1 before scheduling ZOLDR ACCEPT phase 2.

Input

- R6** Parameter list containing the following:
- Total number of #XPRGn records allocated for the program base
 - Number of available #XPRGn ordinals remaining
 - Indicator for whether the master extra program record was read successfully
 - Program base indicator
 - Total number of #APRGn records allocated for the program base
 - Number of available #APRGn ordinals remaining
 - Indicator for whether the master #APRGn record was read successfully.

Programming Considerations

- UELI is an assembler segment.
- Control must be returned to the caller.

Return Values

None.

FILE_CY2KT (UPX7)

The FILE_CY2KT user exit, UPX7, performs accounting or utility functions when the TPF system files a pool section keypoint table (CY2KT). The FILE_CY2KT user exit also provides a mechanism that allows you to supply your own function to convert CY2KT from its current format to pool expansion (PXP) format or to a user-defined format that is compatible with PXP format.

The FILE_CY2KT user exit is called by the FILE_CY2KT function from segment CYH6.

Input

EBXSW0 Contains the index of the data level where CY2KT is to be filed. The index is in the form of data level 0 (D0) to data level F (DF).

CY2KT is in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

Return Values

R6=0 Default return code. If this value is returned, CYH6 converts CY2KT from 32-way loosely coupled pool support format back to PXP format and moves CY2KT into keypoint 9 (CTK9). Use this return code if you code this user exit and do not change CY2KT.

R6=1 If this value is returned, CYH6 does not change CY2KT or copy it to CTK9. Use this return code if you code this user exit to convert CT2KT to PXP format or a user-defined format compatible with PXP format. Your user exit code must move the converted CY2KT to the appropriate fields in CTK9 and file CTK9 to DASD.

Programming Considerations

- This user exit is called by CYH6 when the conversion mode indicator (CY1MD32) in CTK9 is PXP or CONVERTING. If the conversion mode indicator is 32LC or FALLING_BACK, CYH6 does not call this user exit.
- As shipped by IBM, this user exit issues a BACKC macro to return to the caller. The CY2KT on the data level specified by EBXSW0 remains unchanged.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller.
- If R6=0 on return, CY2KT must be on the data level specified by EBXSW0 and in 32-way loosely coupled pool support format.
- If R6=1 on return, the CY2KT on the data level specified by EBXSW0 must be placed in CTK9 and filed to DASD. The CY2KT is still on the data level specified by EBXSW0.

References

See *TPF Database Reference* for more information about pool support.

FILE_STCCR (UPX3)

The FILE_STCCR user exit, UPX3, performs accounting or utility functions when the TPF system files a short-term common control record (STCCR). The FILE_STCCR user exit also provides a mechanism that allows you to supply your own function to convert the STCCR from its current format in processor storage to pool expansion (PXP) format or to a user-defined format that is compatible with PXP format.

The FILE_STCCR user exit is called by the FILE_STCCR function from segment CYH3.

Input

Data Level E Data level E (DE) contains the STCCR to be filed. The STCCR is in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

Return Values

R6=0	Default return code. If this value is returned, CYH3 converts the STCCR from 32-way loosely coupled pool support format to PXP format. Use this return code if you code this user exit and do not change the STCCR on data level E (DE).
R6=1	If this value is returned, CYH3 does not change the STCCR. Use this return code if you code this user exit to convert the STCCR to PXP format or a user-defined format compatible with PXP format. The user exit must file the STCCR on data level E (DE) to DASD. CYH3 performs a BACKC macro to the caller of CYH3.
R6=2	If this value is returned, CYH3 does not change the STCCR. Use this return code if you code this user exit to convert the STCCR to PXP format or a user-defined format compatible with PXP format and you want CYH3 to file the STCCR to DASD. CYH3 files the STCCR to DASD and returns to the function that called it.
EBXSW4=0	If you set the return code to 1 (R6=1), you must set EBXSW4=0 if STCCR was successfully filed to DASD. If the return code is set to 0 or 2, EBXSW4 is set by CYH3.
EBXSW4=2	If you set the return code to 1 (R6=1), you must set EBXSW4=2 if there was an error while filing STCCR to DASD. If the return code is set to 0 or 2, EBXSW4 is set by CYH3.

Programming Considerations

- The FILE_STCCR user exit is called by CYH3 when the format indicator in the STCCR (CY\$32LC in CY\$CON) on data level E (DE) is not set to 32-way loosely coupled pool support format.
- As shipped by IBM, this user exit calls the BACKC macro to return to the caller. The STCCR on data level E (DE) is not changed.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller.
- If you code this user exit and set the return code to 0 (R6=0), the STCCR on data level E (DE) must be in 32-way loosely coupled pool support format.

- If you code this user exit and set the return code to either 1 (R6=1) or 2 (R6=2), the STCCR on data level E (DE) must be in PXP format or a user-defined format compatible with PXP format.

References

See *TPF Database Reference* for more information about pool support.

FIND_CY2KT (UPX6)

The FIND_CY2KT user exit, UPX6, performs accounting or utility functions when the TPF system retrieves a pool section keypoint table (CY2KT). The FIND_CY2KT user exit also provides a mechanism that allows you to supply your own function to convert CY2KT from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

The FIND_CY2KT user exit is called by the FIND_CY2KT function from segment CYH6.

Input

EBXSW0	Contains the index of the data level where CY2KT is to be constructed. The index is in the form of data level 0 (D0) to data level F (DF). The specified data level is free on entry to the user exit.
EBXSW3	Contains the ordinal of CY2KT to be converted.

Return Values

R6=0	Default return code. If this value is returned, CYH6 converts CY2KT from pool expansion (PXP) format to 32-way loosely coupled pool support format. Use this return code if you code this user exit and do not change CY2KT.
R6=1	If this value is returned, CYH6 returns to its caller without changing CY2KT. Use this return code if you code this user exit and your code converts CY2KT from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

Programming Considerations

- This user exit is called by CYH6 when the conversion mode indicator (CY1MD32) in CTK9 is PXP or CONVERTING. If the conversion mode indicator is 32LC or FALLING_BACK, CYH6 does not call this user exit.
- As shipped by IBM, this user exit issues a BACKC macro to return to the caller. The data level specified by EBXSW0 is free.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller.
- If R6=0 on return to CYH6, the data level specified by EBXSW0 must be free.
- If R6=1 on return to CYH6, the requested CY2KT must be on the data level specified by EBXSW0 in 32-way loosely coupled pool support format or in a user-defined format compatible with 32-way loosely coupled pool support format. CY2KT must appear in the same format it would be in if retrieved directly from the #CY2KT fixed file record type.

References

See *TPF Database Reference* for more information about pool support.

FIND_STCCR (UPX2)

The FIND_STCCR user exit, UPX2, performs accounting or utility functions when the TPF system retrieves a short-term common control record (STCCR). The FIND_STCCR user exit also provides a mechanism that allows you to supply your own function to convert the STCCR from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

The FIND_STCCR user exit is called by the FIND_STCCR function from segment CYH2.

Input

EBXSW3	Contains the ordinal of the STCCR that was retrieved (X'00' – X'0C').
Data Level E	Data level E (DE) contains the retrieved STCCR in pool expansion (PXP) format or a user-defined format compatible with PXP format.

Return Values

R6=0	Default return code. If this value is returned, CYH2 converts the STCCR from PXP format to 32-way loosely coupled pool support format. Use this return code if you code this user exit and do not change the STCCR.
R6=1	If this value is returned, CYH2 returns to its caller without changing the STCCR. Use this return code if you code this user exit and your code converts the STCCR from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

Programming Considerations

- This user exit is called by CYH2 when the format indicator in the STCCR (CY\$32LC in CY\$CON) on data level E (DE) is not set to 32-way loosely coupled pool support format.
- As shipped by IBM, this user exit calls the BACKC macro to return to the caller. The STCCR on data level E (DE) is not changed.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller.
- If your user exit sets the return code to 0 (R6=0), the STCCR on data level E (DE) must be in PXP format.
- If your user exit sets the return code to 1 (R6=1), the STCCR on data level E (DE) must be in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

References

See *TPF Database Reference* for more information about pool support.

Get Global Environment Lists

The get global environment lists user exit, UENV, places variables on the global environment list using environment variables that you have provided by coding the setenv function. UENV is called once for each subsystem during restart.

Input

Environment variables that you have coded using the setenv function.

Programming Considerations

- If UENV is part of an ISO-C dynamic load module (DLM) and an external call is added for which a stub does not exist, you must create a stub.
- To create a stub, code the STUB=YES parameter on the SPPBLD entry in SPPGML in SIP.
- UENV gets called during restart only.

Return Values

None.

Loadset History

The loadset history exit, UELL, makes it possible to track changes to loadsets.

UELL is called when the following commands are performed:

- ZOLDR LOAD
- ZOLDR ACTIVATE
- ZOLDR DEACTIVATE
- ZOLDR ACCEPT
- ZOLDR DELETE
- ZOLDR EXCLUDE
- ZOLDR REINCLUDE
- ZOLDR CLEAR

Input

- Caller identifier
- Loadset name (blank padded on right)
- 32-bit CPU mask with targeted CPU bits ON. The mask is filled with 1's if the calling task affects all CPUs (for example, ZOLDR LOAD, ZOLDR CLEAR, and ZOLDR DELETE).
- Pointer to user field in the entry of Loadset Directory (LSD).

Programming Considerations

- UELL is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were on entry.

Return Values

Void.

Log Recovery Error Processing

The transaction log recovery error user exit, CL99, is called whenever log recovery processing detects an error condition. CL99 determines what action is to be taken for the indicated error condition and returns an action code to the caller.

Input

Information to assess the error condition is passed in the entry control block (ECB) work area. The following fields are passed:

EBW009	Processor index of the log that is being recovered.
EBW011	Processor index of the host processor.
EBER01	Error table index as defined in ICRCT equate values.

Programming Considerations

- This exit provides two tables that define actions that are permitted with all of the associated possible error conditions. These tables also indicate which default actions are provided with the released code.
- Two tables are provided: one for errors that occur during recovery of the host processor log, and a second for errors that occur during log takeover (recovery of the log of another processor).
- You must determine what actions are correct for your TPF system. For example, for some error conditions the log will be reinitialized, resulting in the loss of all the data on the recovery log. If you would rather try to correct the log than lose it, you must modify CL99.
- Do not modify the general registers of the caller.

Return Values

You modify the action performed by the TPF system by returning a different action code. The following fields are returned:

EBW009	The unchanged processor index of the log being recovered.
EBW011	The unchanged processor index of the host processor.
EBER01	Error action code.

LU Registration

The logical unit (LU) registration user exit, UARG, allows you to select the local LUs that the TPF system registers with the APPN network. A local LU is any LU that resides in any processor in the loosely coupled complex.

The TPF system starts the registration process when CP-CP sessions become active. UARG is called, for each local LU, during the registration process.

Input

R2 RVT1 address of the TPF LU to be registered.

Programming Considerations

- IBM provides sample code for UARG, which sets the return code to 0 indicating that the LU should be registered.
- If all TPF local LUs have been predefined to the APPN network, you do not need to register them. Code UARG to set return code 2 in this case.
- A TPF local LU must be defined to the APPN network for a remote LU to establish a session with that TPF LU. The TPF local LU must either be predefined or registered.
- Data level D0 must not be modified. All other data levels are available for use by this user exit.
- EBW000–EBW103 and CE1UR1–CE1UR6 must not be modified. EBX000–EBX103 is available for use by this user exit.

Return Values

R7 contains the return code:

- | | |
|---|---|
| 0 | Register this LU |
| 1 | Do not register this LU, but continue the LU registration process |
| 2 | Do not register this LU and end the LU registration process. |

LU 6.2

The session services interface (SSI) routines are called by CMW0 with the CREMC or ENTNC macro to perform session status notifications. These notifications include:

- Session activation (CSXA)
- Conversation allocation (CSXB)
- Session termination (CSXC).

Input

Data macro SH0LL maps the passed parameters starting at location EBW000.

Programming Considerations

1. These exits provide the skeleton to display the values passed from CMW0.
2. You must implement SSI user exits.

MATIP ASCU List

The Mapping of Airline Traffic over Internet Protocol (MATIP) agent set control unit (ASCU) user exit, UMATAS, defines a list of ASCUs to be included in the Open Confirm response to a Session Open request for a Type-A conversational session when the Session Open request does not contain any ASCUs. UMATAS also defines a list of ASCUs to be included in a Session Open request sent by the TPF system to a remote server on some other remote system.

UMATAS is called in CMACMD.

Input

IPaddress	A pointer to the Internet Protocol (IP) address that sent a Session Open request to the TPF system, or a pointer to an IP address that is to receive a Session Open request from the TPF system.
H1H2	An unsigned short integer containing a 2-byte remote ID (H1H2) used to identify the session.
ASCU_size	An unsigned short integer containing a 2-byte field that indicates the number of bytes (either 2 or 4) assigned to each ASCU identifier.
list	A pointer to a char* that you fill in that contains a contiguous list of 2-or 4-byte ASCUs. These ASCUs are to be included in the Open Confirm response to a Session Open request or in a Session Open request.

Programming Considerations

- All data levels must be returned to the caller in the same state that they were on entry.
- Use UMATAS for Type-A conversational sessions only.
- UMATAS is responsible for obtaining the storage needed for the ASCU list.

Return Values

UMATAS returns the number of ASCUs in the ASCU list.

MATIP Assign LNIATA

The Mapping of Airline Traffic over Internet Protocol (MATIP) assign LNIATA user exit, UMATAL, allows you to assign a line number, interchange address, and terminal address (LNIATA) to a specific incoming data message. You can also use this user exit to assign an LNIATA to a data message received by the IP Bridge if you know the original Internet Protocol (IP) address.

UMATAL is called in CMADAT, CMOA, and CRIL.

Input

Iniata	A pointer to an unsigned integer that will contain the LNIATA on return.
session_type	An unsigned integer containing the session type (Type-A conversational, Type-A host-to-host, Type-A, Type B, or IP Bridge).
IPaddress	A pointer to the 4-byte IP address of the message origin.
message_ID	<p>A pointer to the message identifier (ID). The contents of this field depend on the session type indicator:</p> <ul style="list-style-type: none">• If Type-A conversational, the agent set control unit (ASCU) identifier (H1, H2, A1, A2 values from MATIP Session Open request) and terminal address from the data packet.• If Type-A host-to-host, the host identifier (H1, H2 values from the MATIP Session Open) and flow ID from the data packet (if present).• If Type B, the sender high-level designator (HLD) and recipient HLD.• If IP Bridge, the port number on which the message was received.

Programming Considerations

- You must be able to assign the appropriate LNIATA based on information obtained from the original MATIP headers.
- All data levels must be returned to the caller in the same state that they were on entry.
- The length of the message identifier field varies depending on the session type:
 - The Type-A conversational field is 5 bytes.
 - The Type-A host-to-host field is 3 bytes.
 - The Type B field is 4 bytes.
 - The IP Bridge field is 2 bytes.

Return Values

UMATAL returns one of the following values:

0 The LNIATA is identified and returned to the caller.

Negative Number

The LNIATA is not identified and, therefore, is not returned to the caller.

MATIP Flow ID

The Mapping of Airline Traffic over Internet Protocol (MATIP) flow identifier (ID) user exit, UMATFI, allows Societe Internationale de Telecommunications Aeronautiques (SITA) host-to-host communication to allocate message charges to the appropriate host airlines. The byte contents are agreed on an airline-to-airline basis for billing purposes.

UMATFI is called in CMADAT.

Input

H1H2 The host identifier (H1, H2, values from MATIP Session Open). If H1, H2 is not provided, the flow ID and NULL pointers for H1, H2 are passed.

flowID The flow ID from the data packet.

Programming Considerations

All data levels must be returned to the caller in the same state that they were on entry.

Return Values

UMATFI returns one of the following values:

0 Continue the processing of the data packet.

Negative Number

End the processing of the data packet.

MATIP Host Name

The Mapping of Airline Traffic over Internet Protocol (MATIP) host name user exit, UMATCH, allows you to select a remote host or modify the host record data area (user_area in TPF collection support database MATIP_DS) for outbound sessions. You can search the host name table and use the line number, interchange address, and terminal address (LNIATA) to select the remote host.

UMATCH is called in CMACMD.

Input

Iniata	The LNIATA of the remote host.
pHost	A pointer to the start of the host name table.
nHosts	The number of host records configured in the host name table.

Programming Considerations

- All data levels must be returned to the caller in the same state that they were on entry.
- The selected host name must be defined in your Domain Name System (DNS) server so it can be resolved by the gethostbyname function.
- You must provide a table that ends with a zero, of LNIATAs and host names. The following is an example:

```
#define LNIATA_TBL
{{0X00D40103,"HOST1.POK.IBM.COM"},
 {0X00D40104,"HOST2.POK.IBM.COM"},
 {0," "}}
```

Return Values

UMATCH returns one of the following values:

Address	The address of the entry in the host name table that contains the selected host name so message traffic can start.
Null	The host name is not in the host name table. The request to start message traffic exits.

MATIP Router

The Mapping of Airline Traffic over Internet Protocol (MATIP) router user exit, UMATRO, allows messages to be routed to the appropriate application when a terminal address table (WGTA) entry is not used to route the message. This user exit is entered when data is received for either Type-A host-to-host messages or Type-B messages.

UMATRO is called in CMADAT.

Input

session_type	An unsigned integer containing the type of message (Type-A host-to-host or Type B).
sock	An integer containing the socket number.
IPaddress	A pointer to the 4-byte Internet Protocol (IP) address of the message origin.
message_text	A pointer to a data message.
subtype	An integer containing the traffic subtype for Type-A host-to-host sessions, or end-to-end messaging responsibility transfer protocol used for Type B.
message_ID	A pointer to the message identifier. <ul style="list-style-type: none">• For Type-A host-to-host sessions, the message identifier contains the H1, H2, and flow ID.• For Type-B sessions, the message identifier contains the sender and recipient high-level designator (HLD).
pload_len	The data length.

Programming Considerations

- The message passed to UMATRO is not translated or converted to AM0SG format, and an routing control parameter list (RCPL) is not built before UMATRO is entered.
- UMATRO is not used for Type-A conversational sessions.
- Equates for the type of message are defined in the `c$trmq.h` header file, which is included by UMATRO.
- Equates for the traffic subtype for Type-A host-to-host sessions and for end-to-end messaging responsibility transfer protocol used for Type-B sessions are defined in UMATRO:
 - IATA host-to-host traffic subtype for Type-A host-to-host sessions.
 - SITA host-to-host traffic subtype for Type-A host-to-host sessions.
 - BATAP end-to-end message responsibility transfer protocol for Type-B sessions.

Return Values

UMATRO is expected to be entered without a return to the caller.

MATIP Security

The Mapping of Airline Traffic over Internet Protocol (MATIP) security user exit, UMATSE, validates Session Open requests received from a remote system.

UMATSE is called in CMACMD.

Input

IPaddress	A pointer to the 4-byte Internet Protocol (IP) address of the message origin.
session_type	An unsigned integer containing the session type: Type-A conversational, Type-A host-to-host, Type-A, or Type B.
remote_ID	<p>An unsigned short integer containing a 2-byte remote identifier (ID) that is used to help identify the remote host.</p> <ul style="list-style-type: none">• For Type-A conversational, this field contains the number of bytes assigned to each agent set control unit (ASCU) identifier (either 2 or 4 bytes).• For Type-A host-to-host, this field contains the H1H2 (2-bytes that contribute to the identifier) of the remote host.• For Type B, this field contains the high-level designator (HLD) of the sender.
ASCU_in	A pointer to the beginning of a contiguous list of ASCUs that have been sent as part of the Session Open command for Type-A conversational traffic. This field is set only for Type-A conversational sessions and is set to NULL for other sessions.
ASCU_count	An integer indicating the number of ASCUs present in the contiguous list of ASCUs.
ASCU_accept	A pointer to char * that you fill in, which contains a list of ASCUs to accept from the Session Open request for Type-A conversational sessions. This field is set to NULL for all other sessions.
accept_count	A pointer to an integer indicating the number of ASCUs that are being accepted by the user.
ASCU_reject	A pointer to char * that you fill in, which contains a list of ASCUs to be rejected from the Session Open request for Type-A conversational sessions. This field is set to NULL for all other sessions.
reject_count	A pointer to an integer indicating the number of ASCUs that are being rejected by the user.

Programming Considerations

- All data levels must be returned to the caller in the same state that they were on entry.
- Session types are defined in the c\$trmq.h header file and are limited to: TPMATPA, TPMATPH, TPMATPRT, and TPMATPB.

Return Values

UMATSE returns one of the following values:

- 0** The Session Open request is not rejected and control is returned to the caller.

Negative Number

The Session Open request is rejected .

MATIP Session Start

The Mapping of Airline Traffic over Internet Protocol (MATIP) session start user exit, UMATSS, initializes a session with a device or remote system. Characteristics for the session can be defined when a line number, interchange address, and terminal address (LNIATA) and an IP address pointer are passed to UMATSS. UMATSS also enables you to start a Type-A host-to-host session or a Type-B session with a remote system when a terminal address table (WGTA) entry is not associated with the session.

UMATSS is called in CMACMD.

Input

session_type	A pointer to an unsigned integer containing the session type (Type-A conversational, Type-A host-to-host, or Type B).
Iniata	A pointer to the LNIATA being addressed. If the LNIATA is equal to -1, no WGTA entry is associated with the session.
IPaddress	A pointer to the <code>in_addr</code> structure that contains the 4-byte IP address returned by the <code>gethostbyname</code> function.
session_ID	<p>A pointer to the message identifier (ID). The contents of this field will be set by the user and will be consistent with the session type associated with the session.</p> <ul style="list-style-type: none">• For Type-A conversational sessions and Type-A MATIP printers, an H1, H2, A1, A2 value can be defined.• For Type-A host-to-host sessions, an H1, H2, and flow ID can be defined.• For Type B sessions, a sender and recipient high-level designator (HLD) can be defined.
encoding	A pointer to an integer containing the encoding of the format (such as ASCII, EBCDIC, and so on).
subtype	<p>A pointer to an integer containing the agent set control unit (ASCU) length, traffic subtype, or messaging responsibility transfer protocol used.</p> <ul style="list-style-type: none">• The ASCU length for Type-A conversational sessions can be:<ul style="list-style-type: none">– 0 if no ASCUs are being specified– 2 if a 2-byte ASCU is being specified– 4 if a 4-byte ASCU is being specified.• The traffic subtype for Type-A host-to-host sessions can be IATA host-to-host or SITA host-to-host.• The message responsibility transfer protocol for Type-B sessions can be BATAP. The default is routing based on the application index in the associated terminal address table (WGTA) entry.
multiplex	<p>The type of ASCU or host-to-host session:</p> <ul style="list-style-type: none">• The ASCU type for Type-A conversational sessions can be:<ul style="list-style-type: none">– A group of ASCUs with a 4-byte ID for each ASCU (H1H2A1A2)– A group of ASCUs with a 2-byte ID for each ASCU (A1A2)– A single ASCU for a single session (A1A2).• The type of session for Type-A host-to-host can be:

- A multiple flow inside the TCP/IP connection (more than one host can be connected)
- A single flow (a single host connected to a single host).

Programming Considerations

- All data levels must be returned to the caller in the same state that they were on entry.
- You must modify UMATSS to initiate a MATIP session with a device or remote system if you do not use the ZMATP command to associate an IP address with the device or remote system.

Return Values

UMATSS returns one of the following values:

0 The session setup data is returned.

Negative Number

 The session setup data is not returned.

MATIP Translation

The Mapping of Airline Traffic over Internet Protocol (MATIP) translation user exit, UMATTR, translates message text according to user requirements. UMATTR is entered during data packet processing of input or output.

UMATTR is called in CMADAT, CMOA, CMOB.

Input

message	A pointer to the message being sent.
encoding_in	An integer containing the encoding of the message on entry to UMATTR.
encoding_out	An integer containing the encoding of the message on exit from UMATTR.

Programming Considerations

- All data levels must be returned to the caller in the same state that they were on entry.
- For inbound messages, UMATTR will translate from the encoding specified in the original Session Open request to EBCDIC encoding.
- For outbound messages, UMATTR will translate from EBCDIC encoding to the encoding specified in the original Session Open request.
- Equates for encoding are contained in the user exit and are limited to padded Baudot, IPARS, ASCII, and EBCDIC.
- EBW100 to EBW103 contains the message length and should not be written over.

Return Values

- When the translation takes place, a pointer to the translated message is returned to the caller.
- When the translation does not take place, a pointer to the original message is returned to the caller.

Message Queue Interface (MQI) Channel Exits

Standard MQI client support comes with 3 user exits that allow you to customize the channel interface. These exits are referred to as message channel agent (MCA) exit routines in the MQSeries publications. These exits are optional and only called if they have been defined in the MQI channel directory using the ZMQID DEFINE or ZMQID ALTER command. The MCA exits are:

Security	Normally, security exits work in pairs, one at both the client and server ends of the channel. This exit is provided to give the customer the ability to check authorization to start the channel connection. The security exit is called when the MQI channel is first started.
Receive	The receive exit is called after each message segment is received from the server.
Send	The send exit is called just before a message segment is transmitted on the server.

Programming Considerations

1. The interface for the MCA exits are specified in the `cmqxc.h` header file. The user exit code will be given control using the standard ISO-C DLM enter.
2. If an irrecoverable error occurs, the MCA user exits can SERRC with exit. Otherwise, the user exits should always return to the MQI client support code, only manipulating the data passed in the documented interface.
3. Any fields used in the ECB should be restored to their original values before returning to the MQI client support code.

Note: For a detailed description on how the MCA exits work, see the *MQSeries Distributed Queue Management Guide*. For more information about defining or changing MQI channel definition exit attributes see the ZMQID DEFINE and ZMQID ALTER command descriptions in *TPF Operations*.

Message Router

The message router exit, COBC, is called by COA4 before routing the message to the application.

This user exit can be used in the following ways:

1. The text of the message on data level 0 can be altered.
2. The destination of the message can be changed by altering the destination field in the RCPL.
3. Additional information can be passed to the destination application by changing the RCPL to the expanded format and adding the general data area.
4. By inspecting R5 and changing its contents, it is possible to force an I-stream.

Input

R5	Destination I-stream number
D0	Input message in AM0SG format
D1	Reserved
D2	Reserved
D3	Reserved

EBW000–EBWXXX

The RCPL of the destination message (the length depends on whether it is an expanded RCPL and the size of the general data area).

EBX064–EBX103

Reserved

EBROUT Reserved

CE1DBI Database ID of the basic subsystem

CE1PBI Program base ID of the basic subsystem

CE1SSU Subsystem user ID of the basic subsystem

CE1ISN Reserved.

Programming Considerations

- Data level 0 should still contain the input message on return to COA4. Data level 1 through 3 must not be altered. All other levels must not be holding any core blocks.
- COBC is a dynamic exit and is only called when SYSTC switch SBCOMXT is set on.
- During the system installation process, 'COMMEXIT=YES' on the MSGRT macro must be coded to set SBCOMXT on.

Module Copy Selection/Validation

The module copy selection/validation user exit, UCPY, is entered at the point where additional module verification checks can be done. For example, you could add code here to determine if the modules to be copied are on the same channel. Additional user information can be provided to UCPY by entering the ZMCPY ALL and ZMPCY UP commands with the TOKEN parameter specified. See *TPF Operations* for more information about these commands.

UCPY is called by the ZMCPY ALL and ZMPCY UP commands.

Input

ECB data field EBW048–EBW052 contains the user token if the TOKEN parameter is specified when the ZMCPY ALL or ZMPCY UP command is entered. If the TOKEN parameter is not specified, this field is set to X'00'. If the TOKEN parameter is specified, the format of the field is *l**tttt* where:

l is the length of the token.

tttt is the 1- to 4-character token.

Programming Considerations

- IBM ships UCPY in skeleton form with one return statement.
- All data levels, registers, and ECB work areas must be returned to the caller in the same state they were on entry.
- UCPY will issue any output messages that may be associated with validation errors found here.

Return Values

R2 contains the return code:

0 Indicates that the copy will be aborted.

Nonzero Indicates that copy processing can continue.

Nonsocket Activation

The nonsocket activation user exit, CLCQ, allows nonsocket Common Link Access to Workstation (CLAW) applications to be activated. CLCQ is activated during system cycle-up processing if nonsocket CLAW applications exist.

Input

- A pointer to the CLAW device table (CDT) (structure defined in c\$iscddt.h). The structure contains the following information:
 - CLAW adapter ID
 - Workstation name
 - Workstation application name
 - Host application name
 - CLAW symbolic device address (SDA)
 - CLAW device status indicator.

Programming Considerations

If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Nonsocket Connect

The nonsocket connect user exit, CLCU, allows nonsocket Common Link Access to Workstation (CLAW) applications to be activated. CLCS is activated during connect processing.

Input

EBW000	EP ID (X'02')
EBW004–EBW011	Device name
EBW012–EBW015	Pointer to CLAW adapter block
EBW016–EBW019	Global anchor
EBW020–EBW023	Path anchor
EBW024–EBW031	Reserved
EBW032–EBW033	Path ID
EBW034–EBW035	Reserved
EBW036–EBW043	Host application name
EBW044–EBW051	Workstation application name.

Programming Considerations

If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Nonsocket Deactivation

The nonsocket deactivation user exit, CLCM, allows nonsocket Common Link Access to Workstation (CLAW) applications to deactivate nonsocket CLAW application resources. CLCM is activated by CLAW device failures, device disconnects, and during system cycle-down processing if nonsocket CLAW applications exist.

Input

- A pointer to the CLAW device table (CDT) (structure defined in `c$iscddt.h`). The structure contains the following information:
 - CLAW adapter ID
 - Workstation name
 - Workstation application name
 - Host application name
 - CLAW symbolic device address (SDA)
 - CLAW device status indicator.

Programming Considerations

If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Nonsocket Message

The nonsocket message user exit, CLA4, allows messages to be routed to specific nonsocket applications. CLA4 is entered when a nonsocket workstation sends a message to the TPF host.

Input

EBW000	Entry point (EP) ID (X'01')
EBW004–011	Device name
EBW012–015	Adapter ID
EBW016–019	Global anchor
EBW020–023	Path anchor
EBW024–027	Buffer address
EBW028–031	Buffer type and size
EBW032–033	Path ID.

Programming Considerations

- If this user exit is updated to contain writable static, compile it using the RENT option.
- Because socket is currently the only supported Common Link Access to Workstation (CLAW) protocol application, CLA4 is shipped with code that issues a SNAPC dump.

Return Values

None.

Output Message Filtering

Segment UOP1, the first of 2 user exits, is called before segment CVIQ builds its internal routing table and enters segment CVI1 to duplicate the message.

Segment UOP1 allows the routing destination of the message to be modified. If desired, the message can be suppressed by turning on the Message Suppression Indicator (EBX074). The message can also be rerouted by changing the routing code that has been passed to the user exit in Register 6 (R6). The message can be routed to a functional support console (FSC) on your TPF system or on another TPF system.

Note: If the LMT bypass bit (bit CE1ALMT of CE1CPA in the ECB) is ON when UOP1 is entered, do not change the routing code that has been passed to UOP1 in Register 6 (R6).

To reformat a message, alter the text of the message that is in the message block on data level 6. If the message is changed in this exit, all recipients get the changed version. To change the message format for just one FSC destination, place the corresponding routing code in the User Format Indicator (EBX072–EBX073). When the User Format Indicator has been set, UOP2 is entered to enable you to reformat the message for the intended receiver.

The output of the exit is used to build the routing table, and to show whether to invoke the second exit, UOP2.

Note: Since IPL messages are considered critical messages, they are sent to both the PRC and RO CRAS. Messages can only be rerouted after the system has reached 1052 state, or above.

Input

R6 FSC routing codes.

EBX072–EBX073

User format indicator (set to '0' by CVIQ)

EBX074 Message suppression indicator (set to 0 by CVIQ)

D7 Output message block

D9 Mass prefix message block.

Programming Considerations

- Return is NSI in CVIQ on exit from UOP1. CVIQ saves and restores all registers across the exit.
- The released version of UOP1 executes a BACKC macro and returns to CVIQ.
- Performance impact is a user responsibility when implementing user code in UOP1.

Output Message Re-formatting

After the routing table has been built, but before simulation, the second exit, segment UOP2, is called, by CVIQ, if the User Format Indicator has been set. The purpose of this exit is to allow reformatting of the message for just one FSC receiver. You can also provide 3270 simulation code in this user exit, thus choosing to bypass the simulation provided by segment CVIQ. To bypass simulation, the Simulation Indicator (EBX074) that has been passed to the exit, must be turned on. The main use of this second exit is to add special start and end delimiters to a message intended for ISCF/PC without affecting the output to the other consoles.

Input

EBX074	User simulation indicator (set to 0 by CVIQ)
D6	Output message block
D9	Mass prefix message block

Programming Considerations

- Return is NSI in CVIQ on exit from UOP2. CVIQ saves and restores all registers across the exit.
- The released version of UOP2 executes a BACKC macro and returns to CVIQ.
- Performance impact is a user responsibility when implementing user code in UOP2.

Program Event Recording (PER)

The PER user exit, UPER, is called by system error processing. This exit provides a means of changing the output destination for data captured by a PER interrupt. This exit also permits changes in the amount and/or format of the interrupt data displayed.

Input

D0 PER data block

Programming Considerations

1. The identification of the alternate destination must have been specified when PER was activated. The ZSPER command provides the PRINTER parameter for this purpose. The value specified for PRINTER parameter is passed to UPER with the PER interrupt data. The programming required to support this option is a customer responsibility.
2. UPER contains sample code that formats the PER interrupt data and prints it on the RO CRAS.

Program History

The program history exit, UELM, makes it possible to track changes to the program database.

UELME is called by the following commands:

- ZOLDR LOAD
- ZOLDR ACTIVATE
- ZOLDR DEACTIVATE
- ZOLDR ACCEPT
- ZOLDR EXCLUDE
- ZTPLD LOAD
- ZOLDR REINCLUDE
- ZOLDR DELETE
- ZAPGM
- ZAPAT

Note: ZAPGM calls UELM if the file copy of the program is updated. UELM will not be called if the ZAPAT command is entered before E-type loader restart has completed.

Input

- Address of pgm_history_parms structure that contains the parameters available to this function
- D1 - EMR data levels.
- D7 - Contains a block if called from the ZAPGM command.

Programming Considerations

- UELM is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.

Return Values

Void.

REARRANGE_CTK9 (UPX0)

The REARRANGE_CTK9 user exit, UPX0, performs any needed accounting or utility functions when the TPF system retrieves keypoint 9 (CTK9) through the CYYM interface. You can also provide your own function to convert CTK9 from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

The REARRANGE_CTK9 user exit is called by the REARRANGE_CTK9 function in segment CYH0.

Input

EBXSW0	Contains the index of the data level that contains CTK9 in pool expansion (PXP) format. The index is in the form of data level 0 (D0) to data level E (DE).
EBXSW1	Contains switches used by segments CYYM and CYH0. These switches can only be queried by this user exit.

Return Values

R6=0	Default return code. If this value is returned, CYH0 converts CTK9 from PXP format to 32-way loosely coupled pool support format. Use this return code if you code the user exit and do not change CTK9.
R6=1	If this value is returned, CYH0 does not change CTK9 and immediately returns to its caller. Use this return code if you code this user exit and convert CTK9 from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.
R6=2	If this value is returned, CYH0 does not change CTK9. CYH0 runs its error handling function before returning to its caller. Use this return code to cause error processing if you code this user exit and convert CTK9 from its current format to 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

Programming Considerations

- This user exit is called by CYH0 when the conversion mode indicator (CY1MD32) in CTK9 is PXP or CONVERTING. If the conversion mode indicator is 32LC or FALLING_BACK, CYH0 does not call this user exit.
- As shipped by IBM, this user exit issues a BACKC macro to immediately return to the caller. CTK9 remains unchanged.
- You can use all data levels and registers if the data levels and registers are saved before use and restored before returning to the caller. Data level F (DF) contains a work block used by segment CYYM.
- If R6=0 on return, CTK9 must be in PXP format and must be on the same data level where it was on entry to the user exit.
- If R6=1 or R6=2 on return, CTK9 must be on the data level specified by EBXSW0 and in 32-way loosely coupled pool support format or a user-defined format compatible with 32-way loosely coupled pool support format.

References

See *TPF Database Reference* for more information about pool support.

Recoup Command

The recoup command exit, BRU1, will be activated when a ZRECP message has been entered and TPF does not recognize the command. BRPA, the Recoup Command Editor, will enter BRU1 just before issuing the 'ILLEGAL ACTION' message via BRTO.

Input

EBW000-049 Input message. Use MI0MI to map.

R5 Base of input message (MI0MI).

Programming Considerations

1. If control is returned to BRPA, the message is assumed to be invalid, and a message will be issued to that effect.
2. If control is returned to BRPA, then ECB data levels 5 and 6 must be available.

Recoup Phase 1

The recoup chain chase exit, BRU2, is called by BKB1, transfer vector of BKB0. BKB1 is the Phase 1 completion segment. After it has determined that it is not an 'abort' or 'timeout' condition, and that all subsystems have been processed, the exit is activated.

Input

R1=0 Shows entry from BKB1.
R5 Base of global area.
R6 Base of in-core recoup keypoint
R7 Base of recoup save area.
D5 Recoup save area.

Programming Considerations

Control must be returned to BKB1 if user-unique recoup is not activated.

Recoup Restart

The recoup chain chase exit, BRU2, is called by BRSH, the recoup restart segment, if chain chasing was in progress before the system was IPLed. BRSH determines if the RCP tape is open and the sequence number (BK0SQN) is nonzero. If so, BRU2 is entered. Otherwise, recoup is restarted from the beginning.

BRU2 must be able to determine if user chain chasing was in progress. If so, it must determine where it should restart from. If user chain chasing was not in progress, BRU2 must return to allow BRSH to decide where in TPF recoup to restart from.

Input

R1=4 Shows entry from BRSH.
R4 Base of in-core recoup keypoint
R6 Base of global area.
D0 Recoup save area.
D6 Recoup logging block.

Programming Considerations

Control must be returned to BRSH.

Segment URS1

The System Restart exit, URS1, is called by the Restart Scheduler, CTKS, just before CCP restart.

Input

None.

Programming Considerations

1. Control must be returned to CTKS so that restart will complete.
2. URS1 is called only in the BSS. If you need to perform restart on a function in another subsystem, URS1 must be coded to change the database ID (DBI) in the ECB and to invoke user restart segments using the CROSC macro.

Segment URS2

The Restart Schedule exit, URS2, is called by the Restart Scheduler, CTKO, just before reaching 1052 state.

Input

None.

Programming Considerations

1. Control must be returned to CTKO so that restart will complete.
2. URS2 is called only in the BSS. If you need to perform restart on a function in another subsystem, URS2 must be coded to change the DBI in the ECB and to invoke user restart segments using the CROSC macro.

Segment USC1

The Cycle-Down exit, USC1, is activated by the State Change Cycle Down segment, CTKR, just after WGTA keypointing. This exit can be used to keypoint user tables or any other user process during cycle down.

Input

R9 ECB Base

Programming Considerations

Control must be returned to CTKR in order for State Change to complete.

Segment USC2

The Cycle-Up exit, USC2, is activated by the State Change Cycle Up segment, CTKT, just after WGTA keypointing has been activated.

Input

R9 ECB Base

Programming Considerations

Control must be returned to CTKT in order for State Change to complete.

Segment USC3

The Cycle-Down exit, USC3, is activated by the State Change Cycle-Down real-time program, CTKR, at the beginning of each of the following cycle-down schedules:

- from CRAS to 1052
- from MESW to 1052
- from MESW to CRAS
- from NORM to 1052
- from NORM to CRAS
- from NORM to MESW.

Input

R9 ECB Base

Programming Considerations

- Control must be returned to CTKR in order for State Change to complete.
- Check the state (indicated at byte CMMSTI+1) you are cycling from and to before invoking other routines.

Segment USC4

The Cycle-Up exit, USC4, is activated by the State Change Cycle-Up real-time program, CTKT, at the end of the following cycle-up schedules (just before the call to CVCX to set the state indicator):

- from 1052 to UTIL
- from 1052 to CRAS
- from 1052 to MESW
- from 1052 to NORM
- from CRAS to MESW
- from CRAS to NORM
- from MESW to NORM.

Input

R9 ECB Base

Programming Considerations

- Control must be returned to CTKT in order for State Change to complete.
- Check the state (indicated at byte CMMSTI+1) you are cycling from and to before invoking other routines.

Select A Host

The select a host user exit, UAPN, allows you to select the TPF processor in the loosely coupled complex to assign to a new LU-LU session. UAPN is also valuable in a uniprocessor TPF environment because it allows you to select a path (link) for the session.

The TPF system invokes UAPN when a session activation request (LOCATE command) is received on the CP-CP sessions.

Input

R1	RVT1 address of the remote LU
R2	RVT1 address of the local TPF LU
R5	Pointer to the table containing active links for the TPF loosely coupled complex. The ITAPST DSECT maps out the entries in this table.
EBW000	Indicator byte (zero or more bits can be set) X'80' The TPF LU is processor-unique, or the remote LU is an LU 6.2 node that already has one or more sessions with a TPF processor; therefore, the TPF processor has been selected. X'40' The APPN network provided a suggested route for the LU-LU session.
EBW001	The CPU ID of the selected or suggested TPF processor. The value of this field is based on the value of the EBW000 field as follows: <ul style="list-style-type: none">• If EBW000 is 0, EBW001 is not defined.• If EBW000 bit X'80' is on, then EBW001 contains the selected TPF processor and this value cannot be changed by the user exit.• If EBW000 bit X'80' is off and bit X'40' is on, EBW001 contains the suggested TPF processor, which was determined by the suggested link provided by the APPN network. You can choose to not use the selected link and change this value.
EBW010–EBW017	Mode name of the session
EBW020–EBW027	The ALS name of the suggested link. This field is defined only when bit X'40' in field EBW000 is set.

Programming Considerations

- IBM provides sample code for UAPN, which sets the return code in field EBW002 to 0 indicating that the TPF system should select the TPF processor to use for the LU-LU session. A *round-robin* method is used to distribute session activation requests among the active processors in the loosely coupled complex that are in CRAS state or above.
- Data levels D0 and D1 must not be modified. All other data levels are available for use by the user exit.
- If the TPF processor has been chosen (EBW000 bit X'80' is on), you cannot change the selected TPF processor. However, you can select which link to use.

- If the remote LU is the SLU, the APPN network may have included a suggested path in the LOCATE. If so, the name of the suggested ALS, along with the TPF processor connected to that link, are passed as input to this user exit in fields EBW020–EBW027 and EBW001 respectively. You have three choices in this case:
 1. Use the suggested link.
 2. Choose a different link, TPF processor, or both.
 3. Tell the TPF system to select the TPF processor (the APPN network will select).

Return Values

EBW002	Option chosen by the user exit:
0	<p>The user exit did not select a TPF processor or a link. If the TPF system has not already selected the TPF processor (EBW000 bit X'80' is not set), the default logic in the TPF system will be used to select a TPF processor. Regardless of the value of field EBW000, the APPN network will select the link.</p>
1	<p>The user exit selected a TPF processor and, optionally, a link.</p> <p>If the TPF system has not already selected the TPF processor (EBW000 bit X'80' is not set), field EBW001 must contain the CPU ID of the TPF processor you selected. The processor you selected must be in CRAS state or above.</p> <p>If you also selected a link, field EBW020–EBW027 must contain the name of the selected ALS.</p>
2	<p>The user exit chose to use the suggested path. This return code is valid only if EBW000 bit X'40' was set on input to UAPN.</p>
EBW001	CPU ID of the selected TPF processor.
EBW020–EBW027	ALS name of the link selected for the LU-LU session.

Select ALS to Adjacent APPN Node

The select adjacent link station (ALS) to adjacent Advanced Peer-to-Peer Networking (APPN) node user exit, UALS, allows you to select which ALS the TPF system will use to start a session with an LU 6.2 resource that resides in an adjacent APPN node.

UALS is called when the TPF system starts an LU 6.2 session and the local LU is the primary LU (PLU) and the remote LU is known to reside in an APPN node adjacent to the TPF system. Because the remote LU resides in an adjacent node, there is no need to send a LOCATE search on the CP-CP sessions. Instead, a BIND request can be sent directly to the adjacent APPN node. UALS selects the ALS over which to send the BIND request.

Input

- R3** SCB1 address of the LU 6.2 session being started.
- R4** RVT1 address of the adjacent APPN control point (CP).

Programming Considerations

- When mode names are initialized between the TPF system and the remote LU using the change number of sessions (CNOS) procedure, specifying the CP parameter on the CNOSC INITIALIZE macro or ZNCNS INITIALIZE command indicates that the remote LU resides in an adjacent APPN node. Specifying the CP parameter during the CNOS procedure causes UALS to be called when subsequent sessions with this remote LU are activated.
- IBM provides sample code for UALS, which selects the ALS to the specified adjacent APPN CP that has the least number of active LU-LU sessions.
- If there are no active ALS links between the TPF system and the specified adjacent APPN CP (the return code in R1 is set to 0), the TPF system will perform the normal network search to find a path to the remote LU.
See *TPF ACF/SNA Data Communications Reference* for more information about how the TPF system selects the path to use when starting a new LU-LU session.
- EBX048–EBX051 and data levels D0–D15 must not be modified.
EBW000–EBW103, EBX000–EBX047, and EBX052–EBX103 are available for use by this user exit.

Return Values

R1 contains the return code:

- 0 No active ALS exists between the TPF system and the specified adjacent CP.
- x The CCW index of the ALS over which to send the BIND request to start the LU 6.2 session, where x is the CCW index.

Select an RTP Connection

The select a rapid transport protocol (RTP) connection user exit, URTP, allows you to specify which RTP connection to use when an LU-LU session is started. You can start a new RTP connection for the LU-LU session or you can choose an existing RTP connection.

URTP is called only when the following conditions are true:

- An LU-LU session is being started.
- High-performance routing (HPR) support is enabled in the TPF system.
- The primary logical unit (PLU) resides in the TPF system.
- Part or all of the route calculated for this LU-LU session supports HPR support.
- One or more RTP connections that can be used for this LU-LU session already exist.

Input

EBX016–EBX019

Address of the parameter list, which contains the following:

Bytes 0–3	Number of existing RTP connections that can be used for the LU-LU session.						
Bytes 4–n	An 8-byte entry for each RTP connection that can be used, which contains the following: <table><tr><td>Bytes 0–2</td><td>Rapid transport protocol control block (RTPCB) index of the RTP connection.</td></tr><tr><td>Byte 3</td><td>Status of the RTP connection, which can be the following: X'01' Starting X'02' Connected X'03' Switching X'04' Moving X'05' Resynchronizing. See the I RTP_STATUS field in the I RTPB DSECT for more information.</td></tr><tr><td>Bytes 4–7</td><td>Number of LU-LU sessions currently using the RTP connection.</td></tr></table>	Bytes 0–2	Rapid transport protocol control block (RTPCB) index of the RTP connection.	Byte 3	Status of the RTP connection, which can be the following: X'01' Starting X'02' Connected X'03' Switching X'04' Moving X'05' Resynchronizing. See the I RTP_STATUS field in the I RTPB DSECT for more information.	Bytes 4–7	Number of LU-LU sessions currently using the RTP connection.
Bytes 0–2	Rapid transport protocol control block (RTPCB) index of the RTP connection.						
Byte 3	Status of the RTP connection, which can be the following: X'01' Starting X'02' Connected X'03' Switching X'04' Moving X'05' Resynchronizing. See the I RTP_STATUS field in the I RTPB DSECT for more information.						
Bytes 4–7	Number of LU-LU sessions currently using the RTP connection.						

Programming Considerations

- IBM provides sample code for URTP, which selects the RTP connection that has the least number of LU-LU sessions. If all of the RTP connections that can be used have 20 or more LU-LU sessions, the sample code indicates that a new RTP connection should be started.
- The logic for sending output messages processes each RTP connection equally, regardless of the number of LU-LU sessions using each RTP connection. For example, if one RTP connection has 100 LU-LU sessions and another RTP

connection has 5 LU-LU sessions, both of these RTP connections are processed equally. Therefore, try to distribute traffic equally over RTP connections.

- Data levels D0 and D4 must not be modified. All other data levels can be used by this user exit.
- EBW000–EBW103 and EBX000–EBX103 must not be modified.
- See *TPF ACF/SNA Data Communications Reference* for more information about HPR support.

Return Values

R0 contains the return code:

- 0** Start a new RTP connection.
- x** The RTPCB index of the RTP connection to be used.

Select TCP/IP Support

The select TCP/IP support user exit, USOK, allows you to select either TCP/IP offload support or TCP/IP native stack support for a given socket.

USOK is called only when all of the following conditions are true:

- Both TCP/IP offload support and TCP/IP native stack support are defined in the TPF system.
- A socket function call is issued.

USOK is called in segment C536.

Input

applname A character string pointer to the name of the application that issued the socket function.

Programming Considerations

- IBM provides sample code for USOK, that selects offload support for all socket function requests.
- All entry control block (ECB) fields and data levels must be returned to the caller in the same state they were on entry.
- USOK is a subsystem-unique dynamic load module (DLM).
- The value returned by USOK, indicating the type of TCP/IP support, determines how subsequent gethostid and gethostname functions are processed for this ECB. See *TPF Transmission Control Protocol/Internet Protocol* for more information about the gethostid and gethostname functions.
- If the value of **applname** is CLTX, INETD is creating the socket. The server program defined to INETD that is being started is contained in EBX000–EBX003.

Return Values

USOK returns one of the following values:

- 0** Use TCP/IP offload support.
- 1** Use TCP/IP native stack support.

Selective Activate Message Router

The selective activate message router exit, UELH, determines if an ECB is from an ECB origin enabled to use a selectively activated loadset. If it is, UELH copies the list of activation numbers associated with the ECB origin to a core area obtained using MALLOC. It then calls the CEL9 assembler program to set up the ECB field to point to the MALLOC area and update the EAT (ECB Activation Table) slots.

UELH is called by the message router package.

Input

- Length of the activation number.
- Maximum number of selective activation numbers allowed per ECB origin + 1.
- D0 - input message in AM0SG format.
- D1 - May contain AAA.
- D3 - May contain RCB.

Programming Considerations

- UELH is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.
- UELH should acquire MALLOC storage to build the activation number list.
- UELH must call CEL9 to update CE2ANL.
- The high-order bit of the MALLOC storage that contains the address of the activation number list should be set as follows:
 - 0 The order of the selective activation numbers is not significant. The highest activation number assigned to the ECB that has a corresponding program version is used.
 - 1 The order of the selective activation numbers is significant. The first activation number assigned to the ECB that has a corresponding program version is used.
- See “Selective Activate Exits” on page 109.

The CE2ANL field has been added to the ECB. This field will contain the address of the Selective Activation Number List. Opzero initializes CE2ACN with the system's current activation number. UELH initializes CE2ANL based on the origin of the ECB.

Each entry in the CE2ANL list is a 4-byte activation number with the last entry having a F'0' value. The user exit should scan the selective activation index for the ECB origin, and if found, place the associated activation numbers in the list. The maximum number of activation numbers that can be associated with an ECB origin is limited by the size of an SWB (which is used when transferring the activation number list from parent to child ECB).

Notes:

1. The CREM, CRED, and CREX macros pass an ECB activation number from the parent ECB to the child ECB.
2. The CRET and SWISC TYPE-CREATE macros use the current ECB activation number of the system for the child ECB.

The user exit must pass the address of the constructed activation number list to the CE2ANL update routine. IBM code updates CE2ANL because CE2ANL resides in protected page 2 of the ECB which should not be modified by user code.

IBM supplied code (CEL9) will increment the ECB count in the EAT slots associated with each activation number in the activation number list.

Return Values

Void.

Selective Activate Restart

The restart user exit, UELU, is used to rebuild user-defined selective activate tables during an E-type loader restart. UELU is called before calling the task dispatcher to process any entries remaining in the E-type loader control record (ECR) after the ECR is merged with the E-type loader master record (EMR). The UELU exit makes it possible to rebuild any selective activate structures.

The restart user exit, UELU, sets up user defined structures with information about a loadset. This information is used when the loadset is selectively activated.

Note: Enter the ECB origin and loadset names allowed for that origin in a structure.

UELU, is called by:

- An E-type loader restart that reactivates the loadset when:
 - The restart is not from a general file IPL.
 - The subsystem is currently active and valid.
 - There are #OLDx records for this subsystem.
 - EMR can be found and held and ERD records are found without errors.

Input

The EMR data level.

Programming Considerations

- All data levels must be returned to the caller in the same state they were upon entry.
- See “Selective Activate Exits” on page 109.

Return Values

Void.

Selective Activate Structure Initialization

The selective activate structure initialization exit, UELX, makes it possible to initialize selective activate structures.

UELX, is called when the following commands are performed:

- ZOLDR DELETE
- ZOLDR CLEAR

Input

- Caller identifier
- Loadset name (blank padded on right).

Programming Considerations

- UELX is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.
- See “Selective Activate Exits” on page 109.

Return Values

Void.

Selective Activate Structure Update

The selective activate structure update user exit, UELN, is used to set up user defined structures with information about a loadset. This information is used when the loadset is selectively activated.

UELN, is called by:

- The ZOLDR ACTIVATE *lname* SEL command.
- A RESTART that reactivates the loadset.

Input

Loadset name (blank padded on right).

Programming Considerations

- UELN is called by ZOLDR ACTIVATE phase II.
- UELN is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.
- See “Selective Activate Exits” on page 109.

Return Values

Void.

Selective Core Resident Load

The selective core resident load user exit, UCLB, allows you to specify if a specific core resident program will be loaded during restart. UCLB is called by the CLIB segment to determine if a core resident program will be loaded.

Input

R5	Contains the address of the program allocation table (PAT) slot.
CE1DBI	Contains the database identifier (DBI) of the subsystem that is being processed by the CLIB segment.
CE1PBI	Contains the program base identifier (PBI) of the BSS subsystem.

Programming Considerations

If multiple subsystems are present, the CLIB segment loads core resident programs for each subsystem.

Return Values

R2 contains one of the following return codes:

0	Load the program.
Nonzero	Do not load the program.

Selective Recoup

The selective recoup user exit, BRU3, is called by BKA0, the selective recoup start segment, when a descriptor record for the requested ID cannot be found. BRU3 is entered just before the 'UNABLE TO FIND DESCRIPTOR' message being sent via BRTO.

Input

EBW000–019 Input message formatted like a logging block item.

D0 Input message.

Programming Considerations

1. If control is returned to BKA0, then it assumes the input data is incorrect and will issue a message to that effect.
2. If control is returned to BKA0, then ECB data levels 5 and 6 must be available to be used.

SLC Communication Source

The SLC type-B message handler segment (CIM2) is activated by the SLC Communications Source Program (CIMM). CIM2 is only activated when, at system initialization time, you specify that no message switching is to be included in the system.

Input

- R1** Pointer to input message prime block.
- D0** Message block in communication message format (see data macro XM0RL).

Programming Considerations

- CIM2 is shipped in skeleton form. You must implement the functions normally performed by message switching.
- If you have failed to replace this skeleton with the desired message switching package, a system error will result.
- The SLC Communications Source Program (CIMM) enters CIM2 and does not return.

SNA Communication Route Selection

The route selection user exit, CSJV is activated by the CDCINIT request PIU processor (CSJC). This exit selects a virtual route (VR) for an LU-LU session when the TPF system issues the BIND request PIU. The SNA communication route selection exit is necessary because 2 VRs can be active between 2 adjacent subareas. TPF supports VR 0 and VR 1 across channel-to-channel (CTC) links.

Route selection is specified by updating the CCW area index in field RV1CCWIN of the secondary LU's (SLU) resource vector table, part 1 entry. RV1CCWIN equals 0 for VR 0, and 1 for VR 1. CSJV contains sample logic that alternates between VR 0 and VR 1 for session assignment.

Input

D0	Input CDCINIT PIU
EBW060	RID of SLU
EBW064	RVT1 entry of SLU
EBW068	RVT2 entry of SLU
EBW072	SAT entry of SLU.

Programming Considerations

EBW060–EBW075

Must not be modified by exit code.

SNA Message Recovery

The transaction analysis exit routine (CMVU) is activated by the SNA communication source program (CITT) when the system message recovery option is included in the system (the value of the NSRTE parameter of the SIP SNANET macro is greater than 0) and the SNA communication transaction analysis option is requested (by the ZNKEY TRANA-YES command).

You should modify the segment to perform the desired transaction analysis function.

Input

R1	Input message in AM0SG format
R2	RVT1 entry of the SLU
R3	RVT2 entry of the SLU
R4	SRT entry allocated for this message
R6	The time-out factor multiplier defined by the ZNKEY RECIT-m,s command (CK2ITM field in CTK2).
R7	Recovery switch: 1=recover, 0=do not recover. It is set to recover (1) on input.
D0	Input message in AM0SG format.

EBW024–EBW039

Register save area (R1–R4) used by CITT.

Programming Considerations

Registers R1-R4 are saved by CITT in the ECB, and are restored when CMVU returns control to CITT.

SNMP Enterprise-Specific MIB Retrieval

The Simple Network Management Protocol (SNMP) enterprise-specific MIB retrieval user exit, UMIB, provides the TPF system with an interface to retrieve enterprise-specific Management Information Base (MIB) variables. This user exit is called when an SNMP request is received for a MIB variable that is in a group that is not managed by the TPF system.

Input

The UMIB user exit receives a pointer to structure `snmp_struct` with the following fields:

snmp_input_length

An integer indicating the length of **snmp_input_value**.

snmp_input_type

An unsigned character indicating the type of protocol data unit (PDU). The type may be one of the following:

ISNMP_GETREQUEST

Gets the value of the object identifier passed.

ISNMP_GETNEXTREQUEST

Gets the next enterprise-specific MIB variable.

snmp_input_value

A pointer to the object identifier of the variable requested.

Programming Considerations

- The UMIB user exit is responsible for allocating storage for **snmp_output_value**.
- See the `c$snmp.h` header file for the format of `snmp_struct`.
- The **snmp_output_value** can be a maximum length of 255.

Return Values

The UMIB user exit returns one of the following values:

- 0** The enterprise-specific MIB variable is retrieved successfully and the following fields in `snmp_struct` are updated:
- snmp_output_length**
An integer indicating the length of the variable and the value that is pointed to by **snmp_output_value**.
- snmp_output_value**
A pointer to the encoded object identifier followed by the encoded value of that object identifier.
- 1** The enterprise-specific MIB variable does not exist. The SNMP request is rejected with a NOSUCHNAME error return. This is the default return value.

References

See *TPF Transmission Control Protocol/Internet Protocol* for more information about SNMP agent support.

SNMP Manager Validation

The Simple Network Management Protocol (SNMP) request validation user exit, UCOM, validates the identity of the SNMP manager that has sent a request to the SNMP agent in the TPF system.

Input

The UCOM user exit is passed the following values:

ucom_name A character pointer containing the community name received in the SNMP message in EBCDIC format.

ucom_len An integer that contains the length of the community name.

ucom_sockaddr
 A pointer to a sockaddr structure that contains the remote SNMP manager Internet Protocol (IP) address.

Programming Considerations

See the `socket.h` header file for the format of the sockaddr structure.

Return Values

The UCOM user exit returns one of the following values:

- 0** SNMP manager is valid. Processing continues.
- 4** SNMP manager is not valid. AUTHENTICATION_FAILURE trap is sent to all the managers defined in the `/etc/snmp.cfg` SNMP configuration file and the request is discarded.
- 8** SNMP manager is not valid. AUTHENTICATION_FAILURE trap is not sent and the request is discarded. This is the default.

References

See *TPF Transmission Control Protocol/Internet Protocol* for more information about SNMP agent support.

Socket Accept

The socket accept user exit, C542, provides a centralized program to screen all connection requests before they are returned to the server application. C542 is entered each time the accept application programming interface (API) function call receives a connection request from a client.

Input

- The socket descriptor associated with an incoming connection request
- A pointer to the socket address of the connecting client (C structure `sockaddr` defined in the `socket.h` header file)
- The length of the socket address.

Programming Considerations

- The pointer to the socket address that is passed to C542 enables C542 to determine the identity of the connecting client.
- C542 accepts an incoming connection request by returning the socket descriptor passed to it by the accept function back to the accept function.
- C542 rejects an incoming connection request by returning -1 to the accept function.
- If C542 rejects an incoming connection request, the accept function closes the socket descriptor associated with the connection request, sets `sock_errno` to `SOCACCES`, and returns -1 to the server application program that issued the accept function call.

Return Values

- If an incoming connection request is accepted by C542, the socket descriptor passed to it by the accept function is returned to the accept function.
- If an incoming connection request is rejected by C542, -1 is returned to the accept function.

Socket Activation

The socket activation user exit, CLCH, allows you to activate server applications. CLCH is activated when you enter the ZCLAW ACTIVATE command and an TCP/IP offload device is connected to the system or during a system cycle-up.

Input

EBW000–003 Number of Internet Protocol (IP) addresses

EBW004–007 Pointer to the list of IP addresses.

Programming Considerations

- The list of IP addresses and the number of IP addresses are provided to allow you to activate server applications.
- CLCH allows you to build your own IP address table, which can be used by the server applications.
- If you have at least one server for each offload device, use CLCH to activate each server separately.
- If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Socket Connect

The socket connect user exit, CLCS, allows socket Common Link Access to Workstation (CLAW) applications to be activated. CLCS is activated during connect processing.

Input

EBW000	EP ID (X'02')
EBW004–EBW011	Device name
EBW012–EBW015	Pointer to CLAW adapter block
EBW016–EBW019	Global anchor
EBW020–EBW023	Path anchor
EBW024–EBW031	Reserved
EBW032–EBW033	Path ID
EBW034–EBW035	Reserved
EBW036–EBW043	Host application name
EBW044–EBW051	Workstation application name.

Programming Considerations

If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Socket Cycle-Up

The socket cycle-up user exit, CLCV, allows you to activate all of your socket server applications at one time. CLCV is entered once during system cycle-up after all active offload devices are connected to the TPF system.

Input

None.

Programming Considerations

- If you have a server that handles the incoming requests for the entire TCP/IP network, use CLCV to activate all your server applications. However, if you have only one server for each offload device, use the socket activation user exit, CLCH, to activate each server separately. See “Socket Activation” on page 205 for a description of this user exit.
- Application programmers can include server programs that handle incoming messages received from any network interface defined to the TPF system.
- See *TPF Transmission Control Protocol/Internet Protocol* as a guide to help you write your socket server code.

Return Values

None.

Socket Deactivation

The socket deactivation user exit, CLCI, allows socket applications to clean up socket application resources. CLCI is activated by Common Link Access to Workstation (CLAW) device failures, device disconnects, when you enter the ZCLAW INACTIVATE command, or during a system cycle-down.

Input

EBW000–003 Number of active file descriptors.

EBW004–007 Pointer to the list of file descriptors.

Programming Considerations

- The list of file descriptors and the number of active file descriptors are provided to allow you to run user-defined socket application clean-up routines.
- If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

Socket System Error

The socket system error user exit, CLCX, allows you to clean up the socket application resources for an entry control block (ECB). CLCX is activated during system error with exit processing.

Input

EBW000–003 File descriptor.

Programming Considerations

- The file descriptor is provided to allow you to run user-defined socket application clean-up routines for an ECB if you receive a system error.
- If this user exit is updated to contain writable static, compile it using the RENT option.

Return Values

None.

System Error Message

The System Error exit, CPSPD, is called by the CPSP, just before terminal notification of a system error.

Input

EBW000–EBW011	
	DUMMY RCPL
EBW012	ECB format flag
EBSW01	Available for use by this program
EBSW02	Available for use by this program
EBSW03	Available for use by this program
EBCM01	Available for use by this program
EBCM02	Available for use by this program
EBCM03	Available for use by this program
EBER01	Available for use by this program
D0–D5	May contain data
D6	Free and available for use

Programming Considerations

1. Control must be returned to segment CPSP so that system error processing will complete.
2. This exit allows you to provide an alternative to the CHECK DATA CALL SUPERVISOR message.
3. This exit allows for the provision of unique system error recovery/clean-up for user applications.

Tape Display Setup

The tape display user exit, UXTD, is called to set up message displays for tape devices supported by TPF 4.1. A message is displayed until it is changed by another Load Display (LDD) command or until the cartridge is physically unloaded. Therefore, UXTD is called at mount, dismount, and tape switch time (to reset the display if the tape is not unloaded).

Note: UXTD will only be activated if the tape device is supported on TPF 4.1.

The message display is set up in the SETUP subroutine, which can be modified. This should be the only code modified in UXTD. On return from SETUP, the calling routine will issue a Load Display command to display the messages set up by this routine.

UXTD is called by the following segments:

- COSK — Control Program Interface
- COSM — Tape Dismount
- COTI — Tape Initialization
- COTM — Tape Mount
- COTR — Tape Remount
- COTS — Output Tape Switch
- COTT — Input Tape Switch.

Input

The following registers are used by the SETUP subroutine:

- R1** Pointer to LDD data area DSECT
- R5** TSTB/1 (Tape Status Table/Section 1) address
- R6** TSTB/2 (Tape Status Table/Section 2) address
- R7** Return address

The contents of R7 through R15 are expected to be unchanged on return from SETUP.

Data Structures

DSECT LDDAREA maps the data portion of the LDD CCW that will be sent to the tape device. This area consists of the following fields:

- LDDFMT – This single control byte will be set up by the control routine on return from SETUP.
- LDDMSG1 – This is the first 8-byte message area. Subfields are defined for the display generated by the default SETUP routine.
- LDDMSG2 – This is the second 8-byte message area. Subfields are defined for the display generated by the default SETUP routine.

The 2 message areas, LDDMSG1 and LDDMSG2, are each 8 bytes. Both are initialized to zeros when SETUP gets control.

Table 5. The Byte Arrangement for Message Display

Message 1	Message 2	Display
Zeros	Zeros	LDD not issued
Nonzeros	Zeros	Message 1 only
Zeros	Nonzeros	Message 2 only
Nonzeros	Nonzeros	Alternating Message 1/Message 2

Tape Library Validation

The tape library validation user exit, CORU, can be activated by any function to validate a tape volume or change the category of a tape volume.

There are 2 branch vectors for CORU (CT7UOUT and CT7UCHNG). The branch vectors are defined in the TAPEQ macro. The branch vectors must be loaded into R1 before entering CORU. If the value passed in R1 is not valid, an OPR-771 system dump is issued and the ECB is exited.

CORU and its branch vectors do the following user-defined functions:

Perform volume serial number validation

Branch vector CT7UOUT checks a tape volume to make sure it can be used as an output tape. CORU is shipped with code that issues a message if the internal volume serial number and the external volume serial number do not match. If the volsers do not match for a ZTINT request, the operation is ended. If the internal volume serial number is not present, it becomes the same as the external volume serial number. You can add your own code to accept or reject certain categories.

Change tape category

Branch vector CT7UCHNG returns a new library category for a tape volume. CORU is shipped with code that changes the category of a tape to a TPF predefined category default from the TSTB2 field, CPMTDCAT. You can change the predefined defaults or add your own.

Input

- R1 - Branch vector.
- EBTSSW - A value, as defined in TAPEQ, that indicates the calling function.
- EBTCAT - The volume's category value.
- EBTVSN - The volume's internal volume serial number.
- EBTVSNE - The volume's external volume serial number.

Programming Considerations

- CT7UOUT is coded to return a condition of 0, 4, or C. If you want to return a condition of 8, you must write the code.
- CT7UCHNG is coded to return a hexadecimal category value from CPMTDCAT.
- All registers, except R1, must be returned to the caller unchanged. R1 must return a value from the branch vector.

Return Values

From branch vector CT7UOUT, R1 should contain one of the following:

- | | |
|----------|--|
| 0 | Volume valid for output. |
| 4 | Volume mismatch; user defined as not allowed for output. |
| 8 | Volume not valid for output. |
| C | Volume mismatch; user defined as allowed for output. |

From branch vector CT7UCHNG, R1 should contain a hexadecimal category value of 0000–FFFF.

TCP/IP Native Stack Support Accept Connection

The TCP/IP native stack support accept connection user exit, UACC, allows you to verify a remote client connection request.

UACC is called when an `accept` or `activate_on_accept` function call is made for a socket that uses TCP/IP native stack support.

UACC is called in segments C511 and CTSG.

Input

- userexit_s** The file descriptor of a new socket associated with the client connection request.
- userexit_addr** A pointer to the socket address buffer that contains the client address information.
- userexit_addrlen** The size of the socket address buffer.

Programming Considerations

- IBM provides sample code for UACC, which accepts all new client connection requests.
- All entry control block (ECB) fields and data levels must be returned to the caller in the same state they were on entry.
- UACC is a subsystem-unique dynamic load module (DLM).
- If UACC rejects the connection request:
 - The TPF system issues a `close` function call for the new socket.
 - The TPF system returns `-1` with `sock_errno` set to `SOCACCES`.

Return Values

UACC returns one of the following values:

- 1** Reject the connection request.
- userexit_s** Accept the connection request.

TPF File System Initialization

The TPF file system initialization user exit, UBOT, allows you to set up your own initial directories, files, special files, symbolic links, access modes, user IDs, and group IDs. UBOT is called immediately after the root directory and base-level special files have been created by the CBOT segment during file system initialization.

Input

When UBOT is called, UID=0 and GID=1; therefore, this user exit has authority to access or change any file or directory.

Programming Considerations

The table that follows shows the initial file system values that are set up by the TPF file system initialization program (CBOT). These values are shown to give you an idea about how to code the initial values for your own directories, files, special files, symbolic links, access modes, user IDs, and group IDs.

Table 6. Initial File System Values

Name	Description	UID	GID	Access	Maj/Min Device #
/	Root directory	0	1	S_IRWXU S_IRWXG S_IRWXO	N/A
/dev	Special file directory	0	1	S_IRWXU S_IRGRP S_IXGRP S_IROTH S_IXOTH	N/A
/dev/null	Null special file	0	1	S_IRUSR S_IWUSR S_IRGRP S_IWGRP S_IROTH S_IWOTH	80010000
/dev/tpf.ormsg	Output message special file	0	1	S_IWUSR S_IWGRP S_IWOTH	00000000
/dev/tpf.imsf	Input message special file	0	1	S_IRUSR S_IRGRP S_IROTH	00010000
/usr	User file directory	0	1	S_IRWXU S_IRWXG S_IRWXO	N/A

Return Values

The UBOT user exit returns one of the following values to CBOT:

- 0 File system user initialization was completed successfully.
- 1 File system user initialization was not completed successfully.

TPF MQSeries Assign LNIATA

The TPF MQSeries assign LNIATA user exit, CUIW, allows you to convert the remote queue name and the remote queue manager name that is associated with the nonpersistent message into a line number, interchange address, and terminal address (LNIATA). This allows the TPF MQSeries to pass inbound nonpersistent messages to existing nonMQ TPF applications.

Input

The following input is passed to the MQ_assign_Iniata C function:

QName The ReplyToQ name.
QMgrName The ReplyToQmgr name.

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIW into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).
- Before entering CUIW, the TPF MQSeries queue manager converts the message from an MQSeries format to an am0sg format.
- After returning from CUIW, the TPF MQSeries queue manager saves the returned LNIATA into the ECB and marks the WGTAC table entry for this LNIATA as an MQ-type terminal. The TPF MQSeries queue manager then passes the am0sg-formatted message to the TPF message router program, COA4, which sends the message to the nonMQ TPF application. Later, the TPF application can issue a ROUTC to send a message to the remote queue manager and remote queue.

Return Values

LNIATA.

TPF MQSeries Channel Message

The TPF MQSeries channel message user exit (rriCALL_MSGEXIT in segment CUIT) allows you to process a channel message. rriCALL_MSGEXIT is called by the TPF MQSeries queue manager after a message has been retrieved from the transmission queue (sender channel), before a message is put to a destination queue (receiver channel), and when a channel connection is started or ended.

Input

pExitParms	A pointer to the MQCXP data structure in c\$cmqxc.h that contains the channel exit parameters (ExitID and ExitReason).
pChannelDef	A pointer to the MQCD data structure in c\$cmqxc.h that contains the channel definition parameters.
DataLength	A received message indicator. If DataLength contains a nonzero number, a message has been received. When this exit is called, DataLength contains the length of AgentBuffer. This exit must set this field to the length of the data in AgentBuffer that is to proceed.
pAgentBufferLength	The length of the agent buffer.
AgentBuffer	When this exit is called, AgentBuffer contains the transmission queue header (MQXQH in cmqc.h), which includes the message descriptor followed by the message data. The first 8 bytes of the data must not be changed. If the message is to proceed, this exit can do one of the following: <ul style="list-style-type: none">• Leave the contents of the buffer untouched• Change the contents (returning the new length of the data in DataLength, which must not be greater than the length of the agent buffer).

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIT into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).
- A channel must first be defined using the ZMQSC DEF CHL command with the MSGEXIT-YES and MSGDATA parameters specified. If a channel is defined with the MSGEXIT-NO parameter specified, rriCALL_MSGEXIT will not be called.

Return Values

Set one of the following exit response codes in the ExitResponse field in the MQCXP structure in c\$cmqxc.h:

MQXCC_OK

Continue normally.

MQXCC_CLOSE_CHANNEL

Close the channel.

MQXCC_SUPPRESS_EXIT

Suppresses calls to this user exit unless the call is to end the channel connection (ExitReason MQXR_TERM).

MQXCC_SUPPRESS_FUNCTION

Put the message on the dead-letter queue.

Note: Any other value passed in the ExitResponse field will cause the channel to be closed.

Set one of the following exit response codes in the ExitResponse2 field in the MQCXP structure in c\$cmqxc.h:

MQXR2_PUT_WITH_DEF_ACTION

Put with default action. This is only used for a receiver channel and indicates that the user ID of the message is obtained from the default user ID for the channel or the UserIdentifier field in the MQMD structure.

MQXR2_PUT_WITH_MSG_USERID

Put with user identifier. This is only used for a receiver channel and indicates that the user ID of the message is obtained from the UserIdentifier field in the MQMD structure.

MQXR2_PUT_WITH_DEF_USERID

Put with default user identifier. This is only used for a receiver channel and indicates that the user ID of the message is obtained from the default user ID for the channel.

Note: Any other value passed in the ExitResponse2 field will cause the channel to be closed.

TPF MQSeries Channel Message Retry

The TPF MQSeries channel message retry user exit (rriCALL_MREXIT in segment CUIT) allows you to try to put a message to a destination queue if a previous attempt failed. rriCALL_MREXIT is called by the TPF MQSeries queue manager when a channel starts, stops, and when the message channel agent (MCA) is not able to put a message to a destination queue. This exit is only valid for a receiver-type channel.

Input

pExitParms	A pointer to the MQCXP data structure in c\$cmqxc.h that contains the channel exit parameters (ExitID and ExitReason).
pChannelDef	A pointer to the MQCD data structure in c\$cmqxc.h that contains the channel definition parameters.
DataLength	A received message indicator. The length of the message (including the transmission queue header). If DataLength contains a nonzero number, a message has been received.
pAgentBufferLength	The length of the agent buffer.
AgentBuffer	The transmission queue header (MQXQH in cmqc.h), which includes the message descriptor followed by the message data.

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIT into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).
- A channel must first be defined using the ZMQSC DEF CHL command with the MREXIT-YES and MRDATA parameters specified. If a channel is defined with the MREXIT-NO parameter specified, rriCALL_MREXIT will not be called.

Return Values

Set one of the following exit response codes in the ExitResponse field in the MQCXP structure in c\$cmqxc.h:

MQXCC_OK

Continue normally.

MQXCC_CLOSE_CHANNEL

Close the channel.

MQXCC_SUPPRESS_EXIT

Suppresses calls to this user exit unless the call is to end the channel connection (ExitReason MQXR_TERM).

MQXCC_SUPPRESS_FUNCTION

Do not retry the message; put the message on the dead-letter queue.

Note: Any other value passed in the ExitResponse field will cause the channel to be closed.

Set the following exit response code in the ExitResponse2 field in the MQCXP structure in c\$cmqxc.h:

| **MQXR2_USE_EXIT_BUFFER**

| Use the exit buffer. This indicates that any data to be passed is in
| ExitBufferAddr, not AgentBuffer.

| **Note:** Any other value passed in the ExitResponse2 field will cause the channel to
| be closed.
|

TPF MQSeries Channel Security

The TPF MQSeries channel security user exit (rriCALL_SCYEXIT in segment CUIT), provides security protection for data that the TPF MQSeries transfers. This ensures that the resources that the TPF MQSeries queue manager owns and manages are protected from unauthorized access.

rriCALL_SCYEXIT provides support for both sender and receiver channels. Channel security exits at both ends of the channel are given the opportunity to send security messages and to reject or terminate a connection.

rriCALL_SCYEXIT is called:

- After a channel is connected but before sending any messages.
- When a security message is received.
- When a channel connection is ended.

Input

pExitParms	A pointer to the MQCXP data structure in c\$cmqxc.h that contains the channel exit parameters (ExitID and ExitReason).
pChannelDef	A pointer to the MQCD data structure in c\$cmqxc.h that contains the channel definition parameters.
DataLength	A received message indicator. If DataLength contains a nonzero number, a security message has been received.
pAgentBufferLength	The length of a received security message.
AgentBuffer	A pointer to a received security message

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIT into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).
- A channel must first be defined using the ZMQSC DEF CHL command with the SCYEXIT-YES and SCYDATA parameters. If a channel is defined with the SCYEXIT-NO parameter, CUIT will not be called.
- If a security message needs to be sent, do the following:
 - Create a security message in a user-defined malloc area.
 - Specify the buffer address of the user-defined malloc area in the ExitBuffer output parameter.
 - Specify the length of the security message in the ExitBufferLength output parameter.

Return Values

Set one of the following exit response codes in the ExitResponse field in the MQCXP structure in c\$cmqxc.h:

MQXCC_OK

Indicates that the security check is successful.

MQXCC_SEND_SEC_MSG

Send a user-defined security message in response to a request from the other end.

MQXCC_SEND_AND_REQUEST_SEC_MSG

Send a user-defined security message with a request to respond.

MQXCC_SUPPRESS_FUNCTION

Close the channel.

MQXCC_CLOSE_CHANNEL

Close the channel.

Note: Any other value passed in the ExitResponse field will cause the channel to be closed.

TPF MQSeries Convert to Object Handle

The TPF MQSeries convert to object handle user exit, CUIV, allows you to convert a line number, interchange address, and terminal address (LNIATA) into an MQ object handle (Hobj) that is associated with a remote queue manager and remote queue. CUIV is called by the TPF MQSeries queue manager for processing outbound nonpersistent messages if the inbound message did not have a replyto queue manager and replyto queue. If the original inbound message had a replyto queue manager and replyto queue, CUIV is not called because the TPF MQSeries queue manager has saved the object handle for the remote queue and queue manager. CUIV is part of an intercept to ROUTC processing which converts a nonMQ message into MQ message format.

Input

LNIATA.

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIV into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).
- Control must be returned to CUIM.

Return Values

MQHOBJ The object handle of the remote queue.

TPF MQSeries Queue Trigger

The TPF MQSeries queue trigger user exit, CUIR, allows you to activate the appropriate application to process the local queue specified in the trigger message. CUIR is called (unless a process was specified for the local queue definition) when the FIRST parameter is specified with the TRIGTYPE parameter of the ZMQSC ALT QL or ZMQSC DEF QL command. If a process was specified for the local queue definition, the process is called instead of CUIR. See *TPF Operations* for more information about defining a trigger type using the TRIGTYPE parameter and specifying a process using the PROCESS parameter with the ZMQSC ALT QL or ZMQSC DEF QL command.

Input

D0 The address of a block that contains the message queuing message descriptor (MQMD) and message queuing trigger message (MQTM).

Programming Considerations

- Compile (with the C++ compiler) and link-edit CUIR into the dynamic load module (DLM) called CUIR (build script called CUIRBS).
- In a loosely coupled environment, CUIR can be called from more than one processor to process the queue.
- CUIR must exit upon completion.

Return Values

None.

TPF MQSeries Start Queue Manager

The TPF MQSeries start queue manager user exit, CUIA, allows you to start your MQSeries applications immediately after the start or restart of the queue manager.

Input

None.

Programming Considerations

Compile (with the C++ compiler) and link-edit CUIA into the MQSeries dynamic link library (DLL) called CMQU (build script called CMQUBS).

Return Values

None.

Trace-by-Terminal

The trace-by-terminal user exit, CDBUXT, is part of the TPF Assembler Debugger for VisualAge Client and TPF C Debugger for VisualAge Client and allows you to verify additional information in the entry control block (ECB) against a user token field that is stored in the trace entry. The user token data is supplied by the user on the TPF Registration Window. CDBUXT is called from segment CDBTBT, which is the entry point function for CDB0.

Input

itbtenry	The structure that maps the trace entry that will be attached to the active ECB. The definition of the itbtenry structure is contained in the c\$term.h header file.
-----------------	--

Programming Considerations

1. IBM ships this user exit as a C function coded with a return value of TRUE.
2. CDBUXT is an object file that is linked into CDB0, which is a C dynamic load module (DLM). The interface to CDBUXT is a C function call; the user exit code can be written in assembler if the TMSPC and TMSEC macros are used for the corresponding prolog and epilog.

Return Values

TRUE	Attach the trace entry to the ECB.
FALSE	Do not attach the trace entry to the ECB. On return from CDBUXT, CDBTBT continues searching its tables for another trace entry that matches this ECB. CDBUXT is called again if another match is found.

User Command Processor

This exit, UME1, allows you to implement message processing for user defined commands or for the ZDIAG command. UME1 works together with UMET, the user command table.

Input

EBX000–EBX013

User command table entry.

D0

Pointer to the input command in output message (OMSG) format. The input command stored in this area has been converted to uppercase characters and ended by the end-of-message character (X'4E').

CE2CRSMSG

Pointer to a copy of the input command that has not been converted to uppercase characters. The input command stored in this area can contain uppercase, lowercase, or other special characters and ends with a null character (X'00').

Programming Considerations

1. CVAA only passes the command table entry to UME1, the command table is not passed.
2. The released version of UME1 contains sample code which will validate the message, using TPF validation routines, and activate the indicated program. You need only code command table entries in UMET (refer to the prologue).
3. CE2CRSMSG is a 4-byte field located in page 2 of the entry control block (ECB) and defined in data area IEQCE2. (Page 2 of the ECB is the 4 KB of protected storage located 4 KB beyond the start of the ECB.) Use base registers 9 and 11 (R9 and R11) to address this field.

User Data Recovery Copy Support

The user data recovery copy support user exit (UDRS) allows you to use user-defined data recovery copy support. This user exit is activated when you enter the ZFDRS command.

Input

- | | |
|------------------|---|
| D0 | A pointer to the input command in output message (OMSG) format. The input command that is stored in this area was converted to uppercase characters; the command ends with the end-of-message character (X'4E'). |
| CE2CRSMSG | A pointer to a copy of the input command that was not converted to uppercase characters. The input command that is stored in this area can contain uppercase, lowercase, or other special characters; the command ends with a null character (X'00'). |

Programming Considerations

- The released version of UDRS contains sample code that will return an error message to indicate that the requested function is not implemented. You must replace the UDRS user exit with code that supports a data recovery copy function.
- CE2CRSMSG is a 4-byte field that is located in page 2 of the entry control block (ECB) and is defined in data area IEQCE2. (Page 2 of the ECB is the 4 KB of protected storage that is located 4 KB beyond the start of the ECB.) Use base registers 9 and 11 (R9 and R11) to address this field.

Return Values

None.

References

TPF Operations.

User Data Recovery Restore Support

The user data recovery restore support user exit (UDRR) allows you to use user-defined data recovery restore support. This user exit is activated when you enter the ZRDRS command.

Input

- | | |
|------------------|---|
| D0 | A pointer to the input command in output message (OMSG) format. The input command that is stored in this area was converted to uppercase characters; the command ends with the end-of-message character (X'4E'). |
| CE2CRSMSG | A pointer to a copy of the input command that was not converted to uppercase characters. The input command that is stored in this area can contain uppercase, lowercase, or other special characters; the command ends with a null character (X'00'). |

Programming Considerations

- The released version of UDRR contains sample code that will return an error message to indicate that the requested function is not implemented. You must replace the UDRR user exit with code that supports a data recovery restore function.
- CE2CRSMSG is a 4-byte field that is located in page 2 of the entry control block (ECB) and is defined in data area IEQCE2. (Page 2 of the ECB is the 4 KB of protected storage that is located 4 KB beyond the start of the ECB.) Use base registers 9 and 11 (R9 and R11) to address this field.

Return Values

None.

References

TPF Operations.

User Device

The user device user exit, UELC, allows you to access a user-defined input device. UELC allows you to open the device, read the next 4KB record from the device, or close the device.

UELCL is called by the common access device routine (CEL6), when the *ddname* indicates a user-defined device.

Input

- Input device characteristics in C\$IDSINQ (General Data Set Inquire Block)
- The following parameters from C\$IDSUXT (User Exit Definitions)
 - An action code (CEL6_OPEN, CEL6_READ_NEXT_RECORD or CEL6_CLOSE)
 - Data level
 - Pointer to saved common access data (structure defined in tpfiio.h)
 - User field in GDS inquiry block.

Programming Considerations

- UELC is shipped in skeleton form, with one return statement.
- All data levels must be returned to the caller in the same state they were upon entry.
- The data read must be put on the data level that is specified in the input.

Return Values

CEL6_SUCCESS	The operation performed successfully.
CEL6_ERROR	Error performing the operation.
CEL6_END_OF_DEVICE	End of device.

User Global Symbol Table

The user global symbol table user exit, UGST, allows you to define global symbols that do not exist in a program for resolution. These definitions can be used as a valid expression request in both the TPF Assembler Debugger for VisualAge Client and the TPF C Debugger for VisualAge Client.

Input

A user-defined global symbol table. See “Programming Considerations” for more information.

Programming Considerations

- A user-defined global symbol must be defined in two places:
 - The user_ugst table in the ugst.cpp file; for example:

```
const TPF_UDGS user_ugst[] = {
    {"PAT",      ugst_loc_pat,      UGST_LOC_PTR}
    {NULL,      NULL,              UGST_LOC_PTR} };
```

You must also provide the function to resolve any user-defined global symbols defined in the user-defined global symbol table. The resolving function is defined as follows:

```
UDGS_RC func(char *parm, UDGS_RESULT *result);
```

The following example resolves UGST symbol PAT:

```
UDGS_RC ugst_loc_pat (NULL, UDGS_RESULT * result)
{
    result->ptr = cinfc_fast (CINFC_CMMPAT);
    return UDGS_RC_OK;
}
```

- Declare the user-defined global symbol as a variable with the proper data type in the ugstdc.c program so that the TPF C Debugger for VisualAge Client can process the symbol attributes to ensure correct display; for example:

```
struct pat *PAT; /* the variable name must match the symbol name */
```

Array declarations must have at least one item, for example:

```
void* DECB[1]
```

You must compile the ugstdc.c program with the NORENT, NODLL, and TEST(SYM,NOBLOCK,NOLINE,NOPATH,NOHOOK) options.

- The UGST user exit is ended by specifying a NULL pointer in the symbol name field.
- You can have multiple versions of the user global symbol table in the TPF system by loading multiple versions of the table through the online loader. The TPF Assembler Debugger for VisualAge Client uses the application ECB activation number as the basis to select the common symbol table with the same or lower activation number. The TPF C Debugger for VisualAge Client runs in the same ECB as the application, so it will always use the common symbol table based on the activation number of the application ECB.
- You can specify one parameter when using global symbols. The parameter is handled as a character string and the pointer to that character string is passed to the function that resolves the global symbol. The parameter must follow the name of the global symbol and be enclosed in parentheses; for example, DECBNAME(MYDECB), where DECBNAME is the name of the global symbol

and MYDECB is the parameter. If the parameter is to include blank spaces, it must include double quotes (""); for example, DECBNAME("MY PARAMETER").

If the global symbol resolves to UDGS_PTR_LIST, the resolving function is responsible for allocating storage to hold the pointer list. Both the TPF C Debugger for VisualAge Client and the TPF Assembler Debugger for VisualAge Client will call the resolving function with `result->ptrList` set to NULL the first time the resolving function is called. The resolving function is responsible for allocating storage to hold the pointer list and to return the pointer list in `result->ptrList`. Both the TPF C Debugger for VisualAge Client and the TPF Assembler Debugger for VisualAge Client will pass `ptrList` to the resolving function on all subsequent calls.

The resolving function must be aware that when a UGST symbol is entered from the TPF Assembler Debugger for VisualAge Client, the debugger ECB (not the application ECB) is in control. The resolving function must include the correct logic to retrieve the information from the application ECB. See the `usot.cpp`, `ugst.cpp`, and `cgstab.cpp` files for examples of the resolving function.

If the return type for the global symbol is UGST_LOC_PTRLIST, the symbol must be declared as an array of one element in the `xxxxdc.c` file so that TPF C Debugger for VisualAge Client can recognize it as an array of pointers.

Depending on the C structure that is declared for the global symbol, TPF C Debugger for VisualAge Client will use various amounts of ECB heap areas to process the global symbol. Declare the global symbol as `void *` and use XMP map to map to the data that will use the least amount of ECB heap area.

Return Values

None.

User Label Routines

UXTH contains 3 sets of label processing routines. The first, UHLSTART is used to write user header labels to an output tape. The second, UTLSTART is used to write user trailer labels to an output tape. The third, IHLSTART, is used to read user header labels from an input tape. Each of these routines calls the user label subroutines, which are user-modifiable.

The user header label routine calls subroutines UHL1, UHL2, ... UHL8 in order until all 8 are called or until any one of the subroutines fails to set up the user label properly. A valid user header label will start with the characters UHLx where x is the current label number. After verifying the label, UXTH writes the record. Should the write fail, control is passed to COSG and error message COTM161E is sent to your console.

The user trailer label routine calls subroutines UTL1, UTL2, ... UTL8 in order until all 8 are called, or until any one of the subroutines fails to set up the user label properly. A properly set up user trailer label will start with the characters UTLx where x is the current label number. After verifying the label, UXTH writes the record. Should the write fail, control is passed to COSG and error message COTM161E is sent to your console.

The user header label routine for input tapes starts by reading from the input tape. If a tape mark is found, return is made with an indicator saying no user labels were found. If an I/O error occurs, return is made with an error indicator set. If a data record is read, it is checked to make sure it is a properly set up user header label. If not, return is made indicating an invalid label record on the tape. If the record is a valid user header label, subroutine IHL1 is called to perform user processing on the label. This process repeats itself, calling IHL2, IHL3, and others each time a valid user header label is read. If a tape mark is found after reading at least one user header label, control is returned with an indicator saying that user header labels were found. If 8 valid user header labels are read in, and another record is read before finding a tape mark, control is returned with an invalid label indication.

The user label subroutines should not use any of the ECB fields. If temporary storage is needed, a portion of the user label work area, at label ULWAUSR, is set aside for this purpose.

Note: If you use the User Label Routines user exit, you should also use the User Header Label Routines user exit. See "User Header Label" on page 103 for more information.

UXTH is called by the following segments:

COSG UXTH is entered from COSG, when an output tape is mounted, to write user header labels; and, when an output tape is dismounted, to write user trailer labels.

COSF UXTH is entered from COSF so the user header labels on a tape can be read.

Input

On entry to UXTH, certain registers are set up to be used by the user label subroutines. These are:

R2 Pointer to the IBM standard HDR1 and HDR2 labels

R3 Pointer to User Label Work Area (mapped by ITUHL)

- R4** Pointer to the TLMR (Tape Label Mask Record) entry
- R5** Pointer to the TSTB/1 (Tape Status Table/Section 1) entry
- R6** Pointer to the TSTB/2 (Tape Status Table/Section 2) entry
- R7** Return address

Programming Considerations

When UXTH is called to write user header records, the TSTB/3 entry for the tape is not yet set up, therefore, R4 has a pointer to the TLMR record. This requires that the equates in TAPEQ be used to access any fields in the TLMR.

All of the calling program's registers are saved in ULWASAVE on entry. This data area should not be modified for any reason, although, user label subroutines can access these values to restore the pointers provided by the caller. The individual register values are saved in ULWASA0 through ULWASA15 corresponding to R0 through R15.

When the first user label subroutine gets control (UHL1, UTL1 or IHL1) registers R2, R4, R5, and R6 will be set up as listed previously. R3 will point to the base of the User Label Work Area (ULWA). It is recommended that this register not be changed. If it is necessary to use R3, however, the pointer to the ULWA can be restored from CE1CR3. When any other user label subroutines get control, register R3 will once again point to the base of the User Label Work Area. R0, R1, R2, R4, R5, and R6 will all contain whatever values they contained on exit from the previous user label subroutine.

Data Structures

DSECT ITUHL maps a 1005-byte work area that is used by UXTH. The work area is broken up into the following areas:

- User-modifiable area
 - ULWALAB1 through ULWALAB8 correspond to the 8 user label definitions. These areas are to be filled in by the UHL and UTL routines, and will contain user labels read from input tapes for the IHL routines.
 - ULWAUSR is a user work area. This area is provided for temporary storage, since ECB fields are not to be modified by the user label subroutines.
- Register save area
 - ULWASAVE defines a 16 fullword register save area. See Programming Considerations for more details.
 - ULWAREG0 through ULWAREG6 are used for temporary storage of registers by the control routines. These areas must not be modified by the user label subroutines.

User Library Function

The library function exit, UELE, makes it possible to activate and track status on programs that are not entered by standard methods.

UELE is called by:

1. The ZOLDR ACTIVATE command, for all programs that are not C library or TPFDF functions.
2. The E-type loader PAT clean up routine, for programs that do not have other versions and are not C library or TPFDF functions.
3. The ZOLDR DEACTIVATE command for all programs that are deactivated.

Input

- Program name
- Caller identifier.

Programming Considerations

1. UELE is shipped in skeleton form, with one return statement.
2. All data levels must be returned to the caller in the same state they were on entry.

Return Values

Void

User Symbol Override Table

The user symbol override table user exit, USOT, allows you to define global symbols that override symbol definitions in the local symbol table or the common symbol table. For example, you can define symbol D0 in the symbol override table as a pointer to the storage block on data level 0 to override the definition in data macro (DSECT) CPSEQ, which has a value of 0.

The following search order is used in the symbol lookup process:

1. The user symbol override table.
2. The local symbol table or the common symbol table.
3. The user global symbol table.
4. The IBM-provided global symbol table. This table is defined in the cgstab.cpp file, which is found in CUDA.DLL.

Input

A user-defined symbol override table. See “Programming Considerations” for more information.

Programming Considerations

- A user-defined global symbol must be defined in two places:
 - The user_ugst table in the ugst.cpp file. USOT is defined as an array of the TPF_UDGS structure shown in the following example:

```
const TPF_UDGS user_usot[] = {  
    {"D0",    usot_loc_d0,    UGST_LOC_PTR},  
    {"D1",    usot_loc_d1,    UGST_LOC_INDIRECT},  
    {"NULL",  NULL,          UGST_LOC_PTR} },
```

In the previous example, you must provide the ugst_loc_d0 and ugst_loc_d1 functions to resolve symbols D0 and D1. The resolving function is defined as follows:

```
UDGS_RC func(char *parm,  UDGS_RESULT *result);
```

See the usot.cpp, ugst.cpp, and cgstab.cpp files for examples of the resolving function.

- Declare the user-defined global symbol as a variable with the correct data type in the usotdc.c program so that the TPF C Debugger for VisualAge Client can process the symbol attributes to ensure correct display; for example:

```
void *D0; /* the variable name must match the symbol name */  
void *D1;
```

Array declarations must have at least one item, for example:

```
void* DECB[1]
```

You must compile the usotdc.c program with the NORENT, NODLL, and TEST(SYM,NOBLOCK,NOLINE,NOPATH,NOHOOK) options.

- The USOT user exit is ended by specifying a NULL pointer in the symbol name field.
- You can have multiple versions of the user symbol override table in the TPF system by loading multiple versions of the table through the online loader. The TPF Assembler Debugger for VisualAge Client uses the application ECB activation number as the basis to select the common symbol table with the same or lower activation number. The TPF C Debugger for VisualAge Client runs in the

same ECB as the application, so it will always use the common symbol table based on the activation number of the application ECB.

- You can specify one parameter when using global symbols. The parameter is handled as a character string and the pointer to that character string is passed to the function that resolves the global symbol. The parameter must follow the name of the global symbol and be enclosed in parentheses; for example, `DECBNAME(MYDECB)`, where `DECBNAME` is the name of the global symbol and `MYDECB` is the parameter. If the parameter is to include blank spaces, it must include double quotes (""); for example, `DECBNAME("MY PARAMETER")`.

If the global symbol resolves to `UDGS_PTR_LIST`, the resolving function is responsible for allocating storage to hold the pointer list. Both the TPF C Debugger for VisualAge Client and the TPF Assembler Debugger for VisualAge Client will call the resolving function with `result->ptrList` set to `NULL` the first time the resolving function is called. The resolving function is responsible for allocating storage to hold the pointer list and to return the pointer list in `result->ptrList`. Both the TPF C Debugger for VisualAge Client and the TPF Assembler Debugger for VisualAge Client will pass `ptrList` to the resolving function on all subsequent calls.

The resolving function must be aware that when a `USOT` symbol is entered from the TPF Assembler Debugger for VisualAge Client, the debugger ECB (not the application ECB) is in control. The resolving function must include the correct logic to retrieve the information from the application ECB. See the `usot.cpp`, `ugst.cpp`, and `cgstab.cpp` files for examples of the resolving function.

If the return type for the global symbol is `UGST_LOC_PTRLIST`, the symbol must be declared as an array of one element in the `xxxxdc.c` file so that TPF C Debugger for VisualAge Client can recognize it as an array of pointers.

Depending on the C structure that is declared for the global symbol, TPF C Debugger for VisualAge Client will use various amounts of ECB heap areas to process the global symbol. Declare the global symbol as `void *` and use XMP map to map to the data that will use the least amount of ECB heap area.

Return Values

None.

VFA Restart

The VFA restart user exit, CVFX, allows you to examine VFA buffers that are marked as delay-file pending and to set a switch that indicates whether the buffers should be filed or purged. CVFX is activated from VFA restart (CVF2) while cycling the TPF system from restart to 1052 state.

CVFX is only activated when a selected VFA buffer control area (BCA) is marked as delay-file pending and TPF transaction services restart (CLM0) was unable to determine if any TPF locks are still held in an external lock facility (XLF) such as the multi-path lock facility (MPLF) or the limited lock facility (LLF).

Input

R7 Contains the address of the current VFA BCA under examination.

Programming Considerations

- The CVFX segment is shipped in skeleton form with an immediate return to VFA restart.
- The default mode is to always file delay-file pending candidates.
- Registers R1–R7, all user areas, and all data levels must be returned to the calling segment in the same condition as when they were entered.

Return Values

R0 should contain one of the following return codes:

0 File the data buffer.
Nonzero Purge the data buffer from VFA.

Virtual IP Address Processor Deactivation

The virtual IP address (VIPA) processor deactivation user exit, UVIP, allows you to specify if a movable VIPA that is currently owned by a failing processor should be moved to another processor in the complex. When a processor is deactivated, UVIP is called one time for each movable VIPA that meets any of the following conditions:

- The movable VIPA is owned by the deactivated processor and the specified VIPA is not already in the process of being moved.
- The VIPA is owned by a deactivated processor and is moving to an inactive processor.
- The VIPA is being moved to the deactivated processor.

If UVIP is not coded, the VIPA is not moved.

Input

The UVIP user exit requires the following values:

- | | |
|-----------|---|
| R1 | A pointer to a zero-terminated list of 1-byte processor IDs to which the VIPA could be moved. This list includes all active processors that have the VIPA defined. A zero byte indicates the end of the list. |
| R2 | A pointer to a 16-byte area containing the VIPA. For Internet Protocol (IP) Version 4, the VIPA is in the last 4 bytes of the 16-byte area. |

Programming Considerations

- The VIPA will not handle Transmission Control Protocol/Internet Protocol (TCP/IP) traffic until it is moved to another processor by using the UVIP user exit or by using the ZVIPA command with the MOVE parameter specified. If the VIPA is not moved, you can reactivate the deactivated processor to use the VIPA again.
- The EBW and EBX work areas cannot be used in the UVIP user exit.
- Data levels D7 and D8 cannot be modified.

Return Values

R1 contains one of the following return codes:

- | | |
|--------------|--|
| 0 | Do not move the VIPA to another processor. |
| CPUID | The low-order byte contains the CPU ID of the TPF processor to which the VIPA will be moved. |

If a zero is returned, the VIPA is not moved. Otherwise, the system will move the movable VIPA to the active processor ID that is returned from the supplied list. If the exit returns a processor ID that is not valid or is inactive, the VIPA is not moved.

If the processor ID that you select for the UVIP user exit is not from the supplied input list of valid CPU IDs, the VIPA is not moved.

Virtual Reader

The virtual reader exit, UELB, makes it possible to perform the following operations on a virtual reader: OPEN, READ NEXT RECORD, CLOSE.

UELB is called by the common access device routine (CEL6), when the *ddname* indicates the virtual reader.

Input

- R6** Parameter list containing the following:
- Action code
 - Spoolid
 - Data level
 - Saved data area.

Programming Considerations

- IBM provides sample assembler code for UELB. This sample code supports a virtual reader for VM/ESA 1.1. Supporting a virtual reader for any other release of VM can require you to change UELB. If changes are made, the interface (input and output) must remain the same.
- The data read must be put on the data level that is specified in the input.

Return Values

R6 should contain one of the following return codes:

- 0 The operation performed successfully.
- 1 Error performing the operation.
- 2 End of device.

WTOPC Page Control

This exit point allows you to control the size of a WTOPC output page.

Input

R0 Default page size

Programming Considerations

- Return is to the calling programs NSI with the page size set in register 0, R0.
- Do not set the page size larger than 500 lines because the message could exceed the 100 block limit, which could cause an OPR-340 system error.
- Some restrictions apply to register and work area usage, see the program listing for details.
- The released version of UOP3 executes a BACKC macro to return to the caller.

3270 Welcome Screen

The 3270 welcome screen, CSLJ, that is shipped by IBM is sample code that shows how to build a 3270 welcome screen and display it on the appropriate terminal following a logon request.

Input

R1 Address of the ISHLL DSECT.

Programming Considerations

- D0–D7 cannot be modified.
- The following fields in the ECB cannot be modified:
 - EBW020–EBW023
 - EBW028–EBW043
 - EBW076–EBW080.
- To change the 3270 welcome screen, update the 3270 data stream in CSLJ.
- The 3270 welcome screen is displayed when session awareness support is generated for a 3270 logical unit (LU). You can specify session awareness for an LU by using the AWARE= option in the RSC statement. See *TPF ACF/SNA Network Generation* for more information about the RSC statement.

Return Values

None.

Global Area

The global area is a portion of fixed main storage designed to contain application records. The global area permits fast and efficient communication among application programs and between application programs and the control program.

For specific information about accessing globals using C Language Support, see the *TPF C/C++ Language Support User's Guide*.

Terminology

The following glossary of selected terms gives a quick overview of terms used throughout this chapter; it is not intended to be complete and exhaustive. Rather than being listed alphabetically, the terms progress from simple to more complex concepts.

globals.

A generic term referring to any or all of the entities in the area of main storage called the global area or to copies of these entities on DASD.

global record.

A logical collection of data, treated as a single entity, that can reside in the main storage global area and that also resides on DASD as **large** (1055-byte) records (also called **application core-resident records**). As that name implies, global records are typically used by application programs that need convenient access to main storage records. TPF macros provide a variety of efficient services for retrieving, addressing, and filing these records. The records can have various attributes such as keypointability, SSU uniqueness or commonality, and I-stream uniqueness or commonality.

global field.

A subset of certain global records, ranging in size from 1 to 256 bytes in length. Global fields are individually addressable using the GLOBZ macro, and they can share many of the attributes of global records.

global block.

The global records loaded from DASD with an ID of GL. These records contain the data that make up the global fields, and they are the first items loaded into global areas 1 and 3.

global directory.

A series of 8-byte fields at the start of global area 1 and global area 3 that point to the main storage address and file address of global records. These pointers are defined, respectively, by the GL0BA and GL0BY DSECTs.

subsystem user (SSU) unique/common.

In a system with the Multiple Database function (MDBF), global records and fields can be shared among subsystem users (SSUs) in a subsystem or can be unique for each SSU. In other words, an ordinal number used by different SSUs either points to the same common record or to different unique ones.

I-stream unique/shared.

Records in a tightly coupled (TC) system can be declared unique or shared among I-streams. This term applies only to global records (not global fields) and only to main storage copies (not DASD copies).

primary globals.

Globals that reside below the 16MB boundary and are therefore accessible to programs running in 24-bit addressing mode.

extended globals.

Global records that reside in an area of storage defined above the 16MB boundary. The extended global area greatly expands the amount of storage available for globals and relieves constraints on the amount of storage available for data that must be accessible to programs running in 24-bit addressing mode. For each primary global area there is a corresponding extended area. A system can be generated with or without extended globals.

global attribute table (GAT).

Each global directory slot has a corresponding global attribute table entry that identifies the characteristics of the global for example, SSU unique/common or I-stream unique/shared. System programs use a special macro to address this table; application programs do not require access to it.

SSU table.

A table generated during system restart that contains, for each SSU in the system, the address, size, and protection key of each associated primary and extended global area and GAT. The SSU table is organized by I-stream within global area within SSU. (The global area organization is SSU within I-stream within global area).

global storage allocator (GOA).

A #GLOBL fixed-file record that contains the information needed by the Application Core Load program (GOGO) to load all the other #GLOBL records to the global areas (see data macro GO1GO). A GOA must be manually prepared for each SSU as input to the System Test Compiler (STC) so that it becomes part of a pilot tape. The first such record in a chain is called the *prime GOA*.

super GOA.

#GLOBL fixed file records that are used by the GOGO program as an index to find the GOA associated with each I-stream/SSU combination. A super GOA must be manually prepared as input to system test compiler (STC) so that it becomes part of a pilot tape.

global synchronization.

A process that permits changes made to global fields and records to be communicated among 2 or more active I-streams in a loosely coupled (LC) or tightly coupled (TC) system to maintain currency. Synchronization involves offline data record generation, online restart processing, and real-time application interface processing.

system interprocessor global table (SIGT).

In an LC or TC system, a table that stores information necessary for controlling the locking, unlocking, and synchronization of user-specified synchronizable global records or fields.

Structure of the Global Areas

TPF primary globals are located in low main storage between the control program and working storage. The primary global area is divided as follows:

- Global area 1 Protected storage
- Global area 2 Unprotected storage
- Global area 3 Protected storage
- Global area 4 (User defined)

Global areas 1, 2, and 3 are called GL1, GL2, and GL3 in the following text.

An extended global area can optionally be specified in high main storage and accessed in 31-bit addressing mode. For systems with more than 16MB of main storage, the extended global area relieves constraints on main storage availability caused by TPF features like MDBF and tightly coupled support.

Figure 4 on page 246 represents the layout of the global area for a base-only system with a single I-stream, that is, a system with no other subsystems, residing in a uniprocessor. (For a system without extended globals, ignore the top half of the diagram.)

Storage protection limits access to one or more storage locations by preventing writing or reading or both. A storage protection key is an indicator associated with one or more areas of storage. An entry must have a matching key to use a particular area of storage. Thus, an area of storage with a key different from an active entry is protected.

The smallest segment of storage affected by a protection key is 4096 (4KB) bytes. Each global area, therefore, starts on a 4KB boundary and is a multiple of 4KB bytes.

GL2 is assigned the same storage protection key as the active entry control block (ECB). GL1 and GL3 are assigned a different key so that the ECB must issue a macro before any updates can be performed. (Extended global areas have the same protection keys as the respective primary global areas.) After the ECB has completed an update, a second macro must be issued to return the key to its original setting. These 2 macros, GLMOD and FILKW, are discussed in "Using the Global Area by Applications" on page 255.

Global Area 1 (GL1)

The global area begins with the GL1 directory, called GL0BA, which is built as the records are loaded into main storage. The directory consists of a series of pointers. Each pointer in the GL1 directory contains the main storage and file address of a record resident in one of 4 places: GL1, GL2, extended GL1, or extended GL2. A series of data records, usually called **global blocks**, occupy the storage **immediately** following the directory. These global blocks are mapped by the DSECTs in GLOBB through GLOBG. Following the global blocks are resident application records. GL1 is a protected area of main storage.

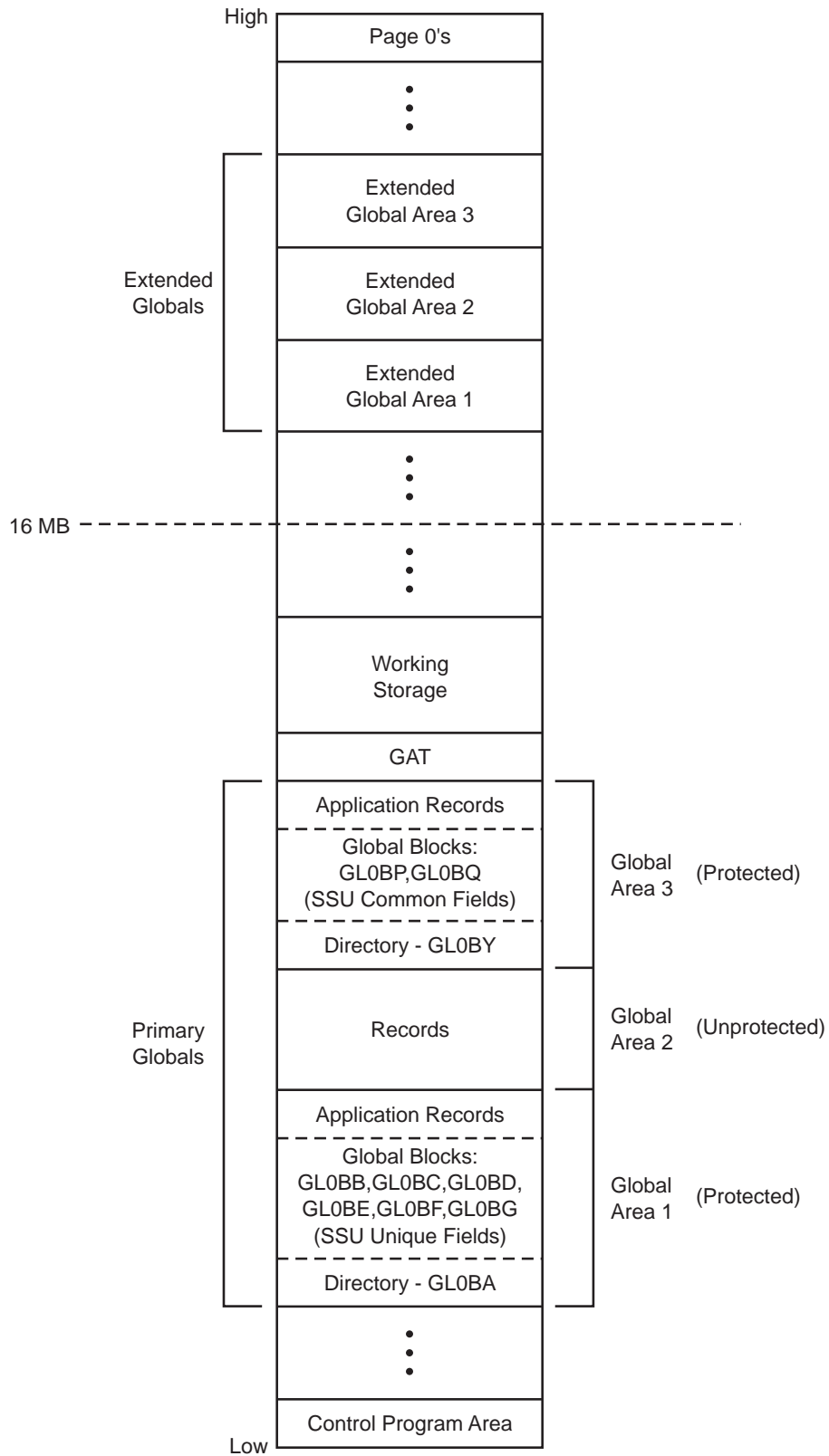


Figure 4. Global Storage Allocation for a Base-Only System with a Single I-Stream

Global Area 2 (GL2)

The GL2 area immediately follows the GL1 area. There is no directory in the GL2 area. The GL1 or GL3 directory has pointers to the records in the GL2 or extended GL2 area.

The GL2 area contains resident application records and special program records. Special program records are concatenations of assembler-produced object decks. They consist of tables, not of executable code. Special program records control the operation of synchronous links. (see *TPF Non-SNA Data Communications Reference* for information about synchronous link control). They are not necessary for all systems. GL2 is an unprotected area of main storage.

Global Area 3 (GL3)

The GL3 area closely resembles the GL1 area. GL3 begins with the GL3 directory, called GL0BY. The structure of GL0BY is the same as GL0BA, a series of pointers. Each pointer in the GL3 directory contains the main storage and file address of a record resident in one of 4 places: GL3, GL2, extended GL3, or extended GL2. Both GL0BA and GL0BY can contain references to records resident in GL2. As in GL1, the directory is followed by a series of global blocks. These are mapped by the DSECTs in GL0BP and GL0BQ. Resident application records in turn follow the global blocks. GL3 is a protected area of main storage.

Global Area 4

TPF allows for the expansion of globals into a fourth area for which there is no general TPF support. For example, the system generation macro GLOBAL provides parameters that describe only global areas 1, 2, and 3 and their extended-global counterparts, although keypoint A and the SSU table provide space for global area 4 pointers. Also, CCCTIN carves space for the global 4 area if CTKA is altered to declare nonzero values for it. Implementation of a fourth global area is strictly the user's responsibility.

Components of the Global Area

This section contains information about the components of the global area.

Global Records

A data record can be thought of as an item stored on DASD or as a block of data in main storage. Global records on DASD are **large** (1055-byte) fixed-file records with the FACE ID #GLOBL. They are defined via the System Test Compiler (STC) and loaded via the command ZSLDR.

The main storage copy of a global record is loaded during system restart. The record can be assigned various attributes such as keypointability, SSU uniqueness or commonality, I-stream uniqueness or commonality.

Certain global records are termed **global blocks**, and are described in "Global Blocks" on page 249. Most of the others are called **application core-resident records**, or simply **core-resident records**.

Still other records stored in the global area, called **special program records**, are special-purpose tables used exclusively by the system.

Keypointable Global Records

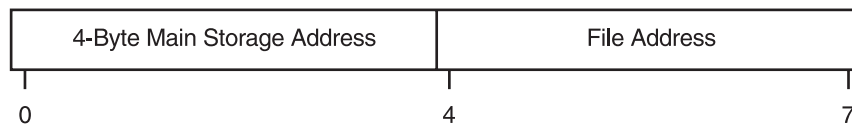
The process of periodically copying a dynamic main storage record to a file on DASD is called **keypointing**. As with most of the control program keypoint records, the records resident in the global area are also maintained on file. Some of these records contain information that rarely changes; these records need not be copied to file. Other records contain temporary information and also need not be copied to file. However, some records contain information that is changeable and must be saved for a restart if a system malfunctions; the file copy must be updated to reflect the changes. These records are classified as keypointable records. Records that do not require dynamic file updating are classified as nonkeypointable records. Global records are defined as keypointable or nonkeypointable in the main storage allocator record (see “GOA List Entry” on page 304.)

Global Directories

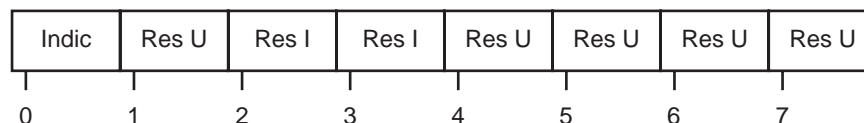
The first 448 bytes of the GL1 area and the first 544 bytes of the GL3 area contain the directories for these areas. Data macro GL0BA defines the GL1 directory as 56 doublewords (448 bytes). The first 48 doubleword slots are for keypointable global records, and the remaining 8 are for nonkeypointable records. The GL0BY data macro defines the GL3 directory as 68 doublewords (544 bytes). The first 64 doubleword slots are for keypointable global records, and the remaining 4 are for nonkeypointable records. Each global area, and therefore each directory, must start on a 4096-byte boundary, and each area must be a multiple of 4096. The first 4096 bytes of GL1 and GL3, including the directory, are directly addressable via the GLOBZ macro.

You can alter the GL0BA and GL0BY data macros and increase the number of slots per directory. However, because GLOBZ provides addressability to a single block of 4096 bytes, care must be taken in increasing the directory size, because the amount of storage remaining for global field use is reduced correspondingly. Also, even if the directory size changes, the number of keypointable slots must remain at 48 for GL1 and 64 for GL3.

These directories are built dynamically with each restart of the system. Each directory slot contains the main storage and the file address of a global block, data record, or special program record as follows:



Each directory slot has a corresponding entry in a special table in the global area called the global attribute table (GAT). The entry contains an indicator byte as follows:



Indic = Indicator byte

Res I= Reserved for IBM use

Res U= Reserved for User

The indicator byte contains information used by system programs during normal online operation, such as whether:

- The global record is keypointable
- A keypointing operation is pending for the global record

- The record resides in unprotected storage
- The record is a special program record in GL2
- The record is SSU common or unique
- The record is I-stream shared or unique
- The record resides in extended storage.

Global Blocks

Global blocks are a series of global records, usually unrelated in content, that are loaded immediately following the global directories. These records contain the global fields described in the next section. Each global block has its own DSECT as follows:

Global Area 1

GL0BA	Directory area
GL0BB	Nonkeypointable miscellaneous fields
GL0BC	Keypointable miscellaneous fields
GL0BD	Fare quote (F/Q) nonkeypointable fields
GL0BE	Fare quote (F/Q) keypointable fields
GL0BF	Expansion for nonkeypointable
GL0BG	Expansion for keypointable

Global Area 3

GL0BY	Directory area
GL0BP	Nonkeypointable TPF global
GL0BQ	Keypointable TPF global

You can create other blocks, for example, GL0BH and GL0BI, to be added to those provided. Extra space must be allocated in the GLOB macro if this is done, and care must be taken that the total space occupied by the directory and the global blocks does not exceed 4096 bytes.

DSECTs GL0BC, GL0BE, GL0BG, and GL0BQ define keypointable fields; GL0BB, GL0BD, GL0BF, and GL0BP define nonkeypointable fields. These DSECTs provide addressability to the global fields and are all pointed to by the macro GLOBZ. Use of GLOBZ and other pertinent macros is explained in “Using the Global Area by Applications” on page 255.

The standard TPF DSECT header is used for global blocks. It contains the following:

- The basic ID, which is the 2-byte string GL
- A record code check byte, which is usually the number of doublewords that are loaded.
- A control byte that shows size and no forward chaining.

The header area can be retained or removed when a nonkeypointable global block is loaded into main storage. Removing the header from a keypointable block causes an error condition during a load sequence. Each global block can individually contain as many as 1056 bytes of data. However, to maintain addressability to the fields in the global blocks, the total length for the directory and the global blocks must not exceed 4096 bytes.

The global blocks are followed by protected data records.

Global Fields

A global field is an addressable unit of storage in a global block. Each of the existing global blocks is designed to contain several global fields and each installation can allocate storage for the global fields as needed.

The rules governing global fields are as follows:

1. Each global field can be from 1 to 256 bytes long.
2. Subsystem user unique fields should be defined only in the GL1 area (MDBF only).
3. Subsystem user common fields can reside only in the GL3 area (MDBF only).
4. Global fields are directly addressable via the GLOBZ macro.
5. Global fields are I-stream unique by default. Each I-stream can only access its own global fields.

Typical coding using global *fields* is:

```
GLOBZ  REGC=R3      Get addressability to GL3
LA      R2,@GLFLD   Field is directly addressed by R2
.                               (This assumes that "@GLFLD" is
.                               defined in GLOBQ or GLOBP)
.
```

Typical coding using global *records* is:

```
GLOBZ  REGR=R3      Get addressability to GL1
L       R4,@GLREC   Pick up address of a global record from directory
.                               (This assumes that "@GLREC" is
.                               defined in GLOBQ)
.
```

Examples of the type of data that can be in the global fields follow:

- System parameters such as recoup restart indicator
- System variables such as time of day and date
- Program switches such as those that are checked during a restart to determine if processing was interrupted and should be continued by any one program
- Application parameters such as the number of days of current inventory

System Environment Considerations

This section contains global area information that is specific for multiple database function and tightly coupled environments.

Multiple Database Function Environment

Multiple Database Function (MDBF) is a licensed feature that supports multiple, discrete applications on the same processor. It consists of 2 distinct yet related concepts: subsystems and subsystem users. Figure 5 on page 251 shows such a system.

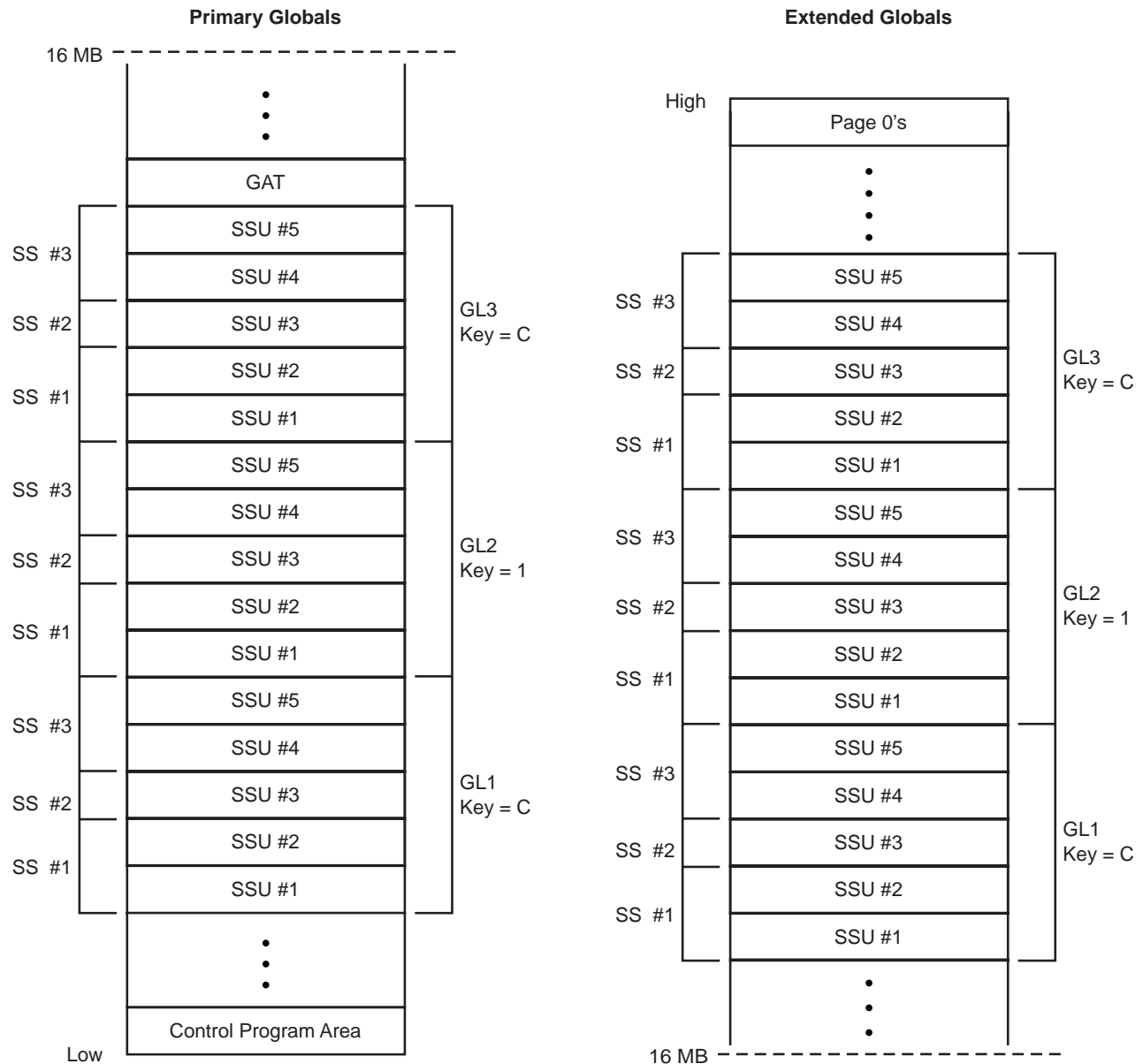


Figure 5. Global Storage Allocation for a Single I-Stream with 2 Subsystems and 5 SSUs

Subsystems Under the Multiple Database Function

A subsystem (SS) can be thought of as an independent application system that shares certain resources with other subsystems. Each subsystem can use all the control program services and can contain a complete base of application programs. For ease of control, one subsystem, called the **basic subsystem** (BSS), contains all the system-related software such as the control program and file-resident support programs. The BSS is the only subsystem that can be hardware IPLed.

The subsystem user (SSU) support allows 2 or more users to use the facilities of the same subsystem. Each SSU can maintain a unique global area, or it can share portions of a common global area with other SSUs.

Globals are among the system resources that are subsystem unique. That is, each subsystem can have its own globals, different from those of any other subsystem. In subsystems, global records can be declared to be common or unique among the SSUs making up the subsystem. Since this is such an important function, special attention is given here to declaring and using SSU common globals.

Example: Given the following assumptions:

- A subsystem has several SSUs
- The GOA entry for GL0BP (a GL3 global block) for each SSU has the X'02' attribute byte set to show GL3 residence
- The GL0BP global block contains the symbolic name @COMMN defining a global field

the following sequence of events occurs:

1. The records defined for the first SSU of this subsystem are physically stored in the GL3 area, and directory entries are generated during the execution of GOGO. (GOGO is the primary loading segment for globals.)
2. When the GOA entry for a record for another SSU is processed, instead of physically loading a GL0BP record, GOGO generates only a directory entry that points to the record loaded for the first SSU.
3. When the application programs of these SSUs need to access the global field @COMMN, the GLOBZ macro with the REGC parameter must be used. This loads the designated register with the GL3 area of the **first** SSU in the subsystem, thus providing addressability to the common field.

For the reasons just given, SSU-common fields **must** be defined in GL3 only. SSU-unique fields can be defined in either GL1 or GL3, but, if they are defined in GL3, they **must** be defined and loaded **before** any SSU-common blocks. Better yet, define them in GL1 only, leaving GL3 for the SSU-common fields.

Note: SSU-common or SSU-unique global records **can** reside in GL1 or GL3.

Design Considerations for Subsystems and SSUs

The following considerations apply:

- An application program of one subsystem cannot access programs, data, or global records that are owned by another subsystem. Nor can it use another subsystem user's unique database or global data.
- The global block DSECTs (for example, GL0BB) for each SSU in a given SS must be identical. The format of the remaining areas can vary from SSU to SSU.
- The control program uses an SSU ID in the ECB to access data and globals. The application program must not alter the SSU ID.
- If a subsystem user becomes large enough, it may need to be separated into a single subsystem. Nothing should be coded at the application level to prevent this separation.
- All SSUs in a subsystem operate in the same system state.

Tightly Coupled Environment

The tightly coupled (TC) environment supports multiple I-streams in each central processing complex (CPC). One of the I-streams in each CPC serves as the main I-stream, while all other I-streams in the CPC are known as application I-streams. The main I-stream services I/O requests, interrupts, and some control program (CP) services. The application I-streams execute applications and some CP services.

All I-streams in the CPC have the same MDBF configuration. To put it simply, all subsystems (SSs) and subsystem users (SSUs) that exist for one I-stream also exist for the other I-streams.

I-Stream Unique/Shared Globals

The implementation of globals for tightly coupled systems allows great flexibility in the allocation of the global area. Since each I-stream has its own global area, each I-stream can maintain a set of I-stream-unique (ISU) global records that are unique to that I-stream and not addressable from other I-streams. ISU records can be accessed only by the owning I-stream.

I-stream shared (ISS) global records that are shared by all I-streams in a CPC can also be defined. Only global **records** can be I-stream shared, not global **fields**. I-stream sharing is allowed in the SSU operating mode for SSU unique records, or in the SS operating mode for SSU shared records. ISS records can be read by any I-stream.

Note: The application **MUST** provide its own serialization method for updating ISS records.

See “Loading Globals” on page 261 for information on loading ISU and ISS records.

Figure 6 on page 254 represents the layout of the global areas for a tightly coupled system with 3 I-streams and 2 subsystem users. Unique page and segment tables provide each I-stream with its own view of the global areas.

The main I-stream’s view of storage is identical to real memory. Real storage below the 16MB boundary is reserved for an area to contain the I-stream shared global records and for one copy of the I-stream unique globals and global directories. Real storage in high memory is reserved for $(n-1)$ copies of the I-stream unique globals and global directories, where n is the number of I-streams.

Each application I-stream sees its I-stream-unique globals in the unique area below 16MB, even though these globals actually reside in high real memory. Thus, applications dispatched on any I-stream are able to access all of the shared and unique globals they need using 24-bit addresses.

The layout of the extended global areas is unchanged from previous releases of the TPF system. Extended globals are allocated subsystem user in I-stream within global area.

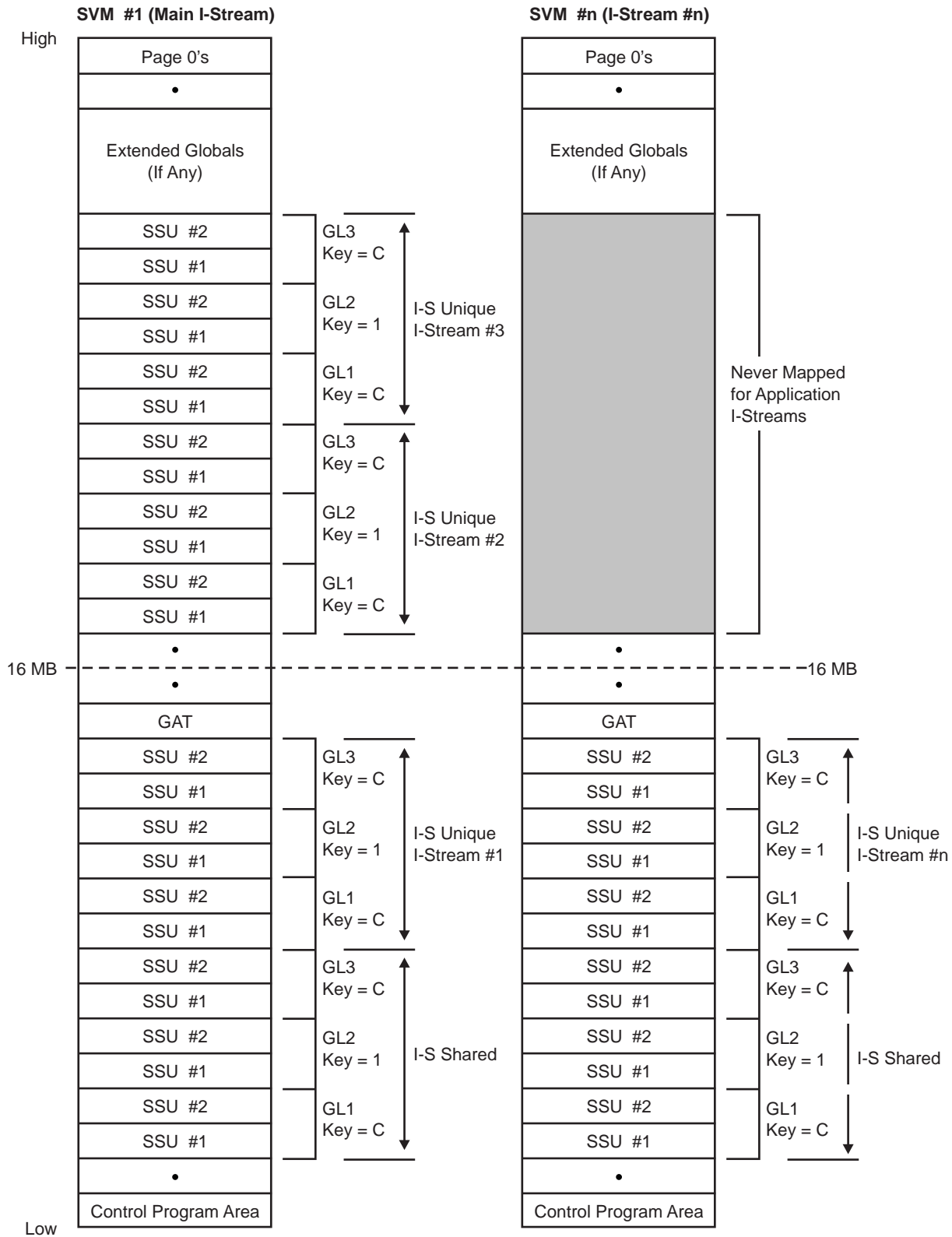


Figure 6. Global Storage Allocation for 3 I-Streams with 1 Subsystem (the BSS) and 2 SSUs

Extended Globals

Extended globals contain global records. They do not contain global directories or global blocks. The extended global area resides above the 16M line and is divided into 3 sections: GL1, GL2, and GL3. There are further sub-divisions in each of these sections. For example, the GL1 area contains a section for each I-stream in the processor, and each of these sections contain a section for each SSU in the complex. There is no separation between I-stream unique and I-stream shared globals. They are both in the same section. The actual definition of the global record (I-stream unique or I-stream shared) is implemented with the pointers to these records which reside below the 16M line inside of the global directories (GLOBA and GLOBY).

Note: Unlike primary globals below the 16M boundary, each I-stream has the ability to access extended globals of other I-streams.

Notes:

1. The virtual address of the beginning of the I-stream-shared and I-stream-unique areas is the same on all I-streams. The overall size of an I-stream unique area is the same for all I-streams.
2. The I-stream-unique primary globals in high memory are all visible on the main I-stream. To ensure that certain system functions, such as critical record filing and ZDCOR/ZACOR, can still access all of main storage.
3. The global directories reside in the I-stream unique areas. Addresses in the global directories can be used only on the I-stream to which the directories correspond.

Note: The addresses in the directories in high memory for I-streams 2– n have no meaning on the main I-stream.

4. It follows that application utility programs should not attempt to access an I-stream-unique global on another I-stream without first switching control to the other I-stream (via the SWISC macro). Also, application programs should not pass the address of a global on one I-stream to a program on another I-stream, since the receiving program may have a totally different view of global storage.

Synchronizing Globals in the Tightly Coupled Environment

Sometimes the same global can be loaded into the global area of each I-stream, for example, data residing in a field that can't be I-stream shared, giving it the appearance of an ISU global. Some of these ISU globals that get updated by an application on one I-stream may need to have their updates transmitted to the other I-streams in the CPC. This process is called global synchronization.

Using the Global Area by Applications

The global area is designed primarily to be used by applications, although the control program does access it. Application programs that access or update the global area must meet certain requirements. Several application programming macros are available to the application programmer to satisfy these requirements. The examples presented in this section are only illustrations; they are not the only solution to a coding problem. A complete description for each macro can be found in *TPF General Macros*.

GLOBZ: Define Global Fields Macro

The application macro GLOBZ allows application programs to access the global area by defining the global fields in the global blocks of GL1 or GL3 or both. A maximum of 4096 bytes can be defined by GLOBZ for each global area. Each field

in the 4096 bytes has a unique symbolic label and can be directly referenced after issuing GLOBZ. GLOBZ does the following:

- When coded with the REGR parameter, GLOBZ calls the storage-reserving macro GLOB, which in turn calls the GL1 directory DSECT macro GLOBA and global field data macros GLOBB, GLOBC, GLOBD, GLOBE, GLOBF, and GLOBG.
- When coded with either the REGS or REGC parameter, GLOBZ calls the GL3 directory DSECT GLOBY and the global field data macros GLOBP and GLOBQ.
- GLOBZ loads one or 2 base registers with the address or addresses of the global areas.

The following 3 examples show how the GLOBZ macro can be coded:

Example 1

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7...
LABEL    GLOBZ REGR=R1      Load a base register with the global
                             area address. In this case, the
                             REGR=R1 specifies GL1. REGS=xxx
                             specifies the GL3 base, where xxx
                             is some base register. REGC=xxx
                             specifies the "common" global fields
                             base (in MDBF), where xxx is some base
                             register.
```

Example 2

A FLD=xxxxxxx parameter may be used to avoid the necessity of knowing in which global area the field is stored. The appropriate address of either GLOBA or GLOBY is returned in the specified register. Either REGR or REGS can be specified with FLD, but not both.

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7...
LABEL    GLOBZ REGR=R1, FLD=@AFIELD
```

Example 3

GLOBZ does not define the fields of the resident data records in GL1, GL2, or GL3. In this case, each resident record type has its own data macro, and the program addresses it by loading a base register with the pointer from the appropriate directory.

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7...
LABEL    GLOBZ REGR=R1
          EX0AU REG=RG2      Assigns register 2 as base for record
                              using data macro that defines record.
          L      RG2,@EXAU   Loads record base with pointer from
                              global directory.
```

Note: Loosely coupled (HPO) and tightly coupled (TC) users use the SYNCC macro to update the global area, because global synchronization is required between I-streams.

GLMOD: Change Global Protect Key Macro

All records in the GL1 or GL3 areas reside in an area of real main storage with a protection key different from that in the program status word (PSW) for application programs. Before an application program can modify any field in these areas, the GLMOD macro must be issued. GLMOD changes the protection key in the PSW of the application program to match the key of the global area.

The following example shows how to change the protection key in an area of real storage:

```

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
LABEL      GLOBZ REGR=R1
.
.
.
LABEL      GLMOD GLOBAL1      Change the global area storage
                                protection key. 'GLOBAL1' is the
                                default and need not be coded.

```

FILKW: File Keyword Macro

The FILKW macro allows application programs to request an update on the file copy of a keypoint record. FILKW is also used to restore the storage protection key to the application working storage key after an update has been made. In the following example, assume GLOBZ contains the field identified as @AFIELD:

```

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
LABEL      GLOBZ REGR=R1
.
.
.
LABEL      GLMOD
            ST      RG5,@AFIELD      Update @AFIELD.
LABEL      FILKW R,@GLBCC      Restore the protection key and update
                                the global record, the main storage
                                address of which is located in the
                                directory at symbolic label @GLBCC.
                                The protection key must be restored
                                before the program can modify "native"
                                areas.

```

GLOBA: Define Global 1 Macro

The GLOBA macro has 3 functions:

- To map out the contents of the GL1 directory. This directory contains main storage and file addresses for the special program records, global blocks GLOBB through GLOBG, and miscellaneous router control data records.
- To set up in a specified register the displacement in the global area 1 directory of the first slot assigned to a special program record.
- To set up an 8-byte field containing 1 bit for each of the 48 keypointable record slots in GL1.

The allocation of storage for GL1 should be reflected in the macro source statements set by each installation.

GLOBY: Define Global 3 Macro

The GLOBY macro has 3 functions:

- To map out the contents of the GL3 directory. This directory contains main storage and file addresses for the special program records, GLOBP and GLOBQ global blocks, and miscellaneous router control data records.
- To set up in a specified register the displacement in the global area 3 directory of the first slot assigned to a special program record.
- To set up an 8-byte field containing 1 bit for each of the 64 keypointable record slots in GL3.

The allocation of storage for GL3 should be reflected in the macro source statements set by each installation.

SYNCC: The Global Synchronization Macro

The SYNCC macro is designed to coordinate the synchronization of global area data among several active I-streams and CPCs in a TC or LC environment. The 3 SYNCC macro options are:

- LOCK, which reserves the exclusive use of a global record or field and refreshes the main storage copy with the most current file copy. The LOCK option also sets the protection key of the user to allow a global update.
- UNLOCK, which releases the exclusive use of a global record and resets the protection key of the user. The UNLOCK option allows a user to release exclusive use of a record without doing an update first.
- SYNC, which:
 - Writes the updated global back out to DASD.
 - Requests synchronization of the global among all I-streams.
 - Releases exclusive use of the global.
 - Resets the protection key of the user.

The FILKW and GLMOD macros should be used to update and file nonsynchronized global fields and records, which are not propagated to other active processors. The FILKW and GLMOD macros should not be used to update synchronized global fields or records; the SYNCC macro must be used to propagate the fields or records to the other active processors and create the main storage images at IPL time. SYNCC macro coding examples can be found in “Examples of Coding the SYNCC Macro” on page 317.

Programs that process both synchronized and nonsynchronized global fields and records must be careful to use the appropriate macros for the 2 kinds of globals.

Programming Considerations

This section contains programming considerations for defining and using global areas.

Defining Addressability to Globals

This section contains programming considerations for defining global areas.

Global Records

When defining a new global, you must decide which global area the new global should be addressed from. A global record can reside in GL1, GL2, or GL3, but it is addressed from a directory slot in GL1 or GL3. To address the new global record from GL1's directory, you must label one of the 56 doubleword slots in the GLOBA macro. The tag on the 8-byte field then points to the GL1 directory slot for the new global record. To address the global record from GL3's directory, you must update the GLOBY macro in a similar manner. If either GLOBA or GLOBY is updated, then the GLOBZ macro, which allows application programs to access the global area, must also be updated. The GLOBZ macro must be updated because it contains a list of global records addressed by GLOBY and GLOBA, and the directory tag for the new GL3 or GL1 global record must be added to that list.

Another consideration is the degree of storage protection desired for the new global. The GL1 and GL3 areas both have protection keys different from that assigned to E-type programs and require the use of GLMOD and FILKW (or KEYRC) macros to change from one key to another. The GL2 area, on the other hand, has the same protection key as E-type programs and can be directly addressed without special macros.

Addressing extended globals presents no special problems: the GL1 directory is used to address records stored in extended GL1 or GL2; the GL3 directory is used to address records stored in extended GL2 or GL3. The chief consideration is that programs written to access extended globals must be allocated to run in 31-bit mode or must switch to 31-bit mode by using MODEC MODE=31.

See “Loading Globals” on page 261 for more information about allocating a new global record and creating an input data set used for the pilot tape.

Global Fields

Whether a global field is to be SSU common or unique determines in which global area the global field should be defined. SSU unique fields should be defined only in the GL1 area. SSU user common fields must reside in the GL3 area.

Global Fields in Global Area 1: To address GL1 records and fields, the GLOBZ macro must be used with the REGR option. This invokes macro GLOB which, in turn, invokes macros GL0BA, GL0BB, GL0BC, GL0BD, GL0BE, GL0BF, and GL0BG.

The following summarizes the result of these macro calls:

1. Besides calling the other macros, GLOB declares storage for the GL1 directory and for the global blocks. GLOB includes the following code:

```
@GLOBA DS 56D      DEFINE THE AREA FOR 56 SLOTS
@GLOBB DS 132D      X'84'          1056 BYTES
@GLOBC DS 132D      X'84'          1056 BYTES
@GLOBD DS 9D        X'09'          72 BYTES
@GLOBE DS 8D        X'08'          64 BYTES
@GLOBF DS 17D       X'11'          136 BYTES
@GLOBG DS 3D        X'03'          24 BYTES

GLOBA      DIRECTORY AREA
GLOBB      NON-KEYPOINTABLE MISC FIELDS
GLOBC      KEYPOINTABLE MISC FIELDS
GLOBD      F/Q NON-KEYPOINTABLE FIELDS
GLOBE      F/Q KEYPOINTABLE FIELDS
GLOBF      EXPANSION FOR NON-KEYPOINTABLE
GLOBG      EXPANSION FOR KEYPOINTABLE
```

2. The GL0BA macro maps the GL1 directory. Storage is declared for 56 doubleword slots. Each slot to which a global will be loaded should have a label. The GL0BA DSECT must not address more directory slots than are declared in the @GLOBA DS instruction.
3. The amount of storage GLOB declares for each global block must be greater than or equal to the amount of storage defined in each DSECT. In the preceding example, the sum of all the global fields defined in GL0BD must be less than or equal to 72 bytes.
4. The sum of the storage declared by GLOB must not exceed 4KB, since the directory and global blocks are all addressed from one base register.
5. The amount of storage declared for each global block must exactly match the number of doublewords that will be loaded into that area (see *Number of Doublewords* under “GOA List Entry” on page 304).

To define addressability to a new global field in GL1, you must modify one of GL1's global field DSECTs: GL0BB, GL0BC, GL0BD, GL0BE, GL0BF, or GL0BG. If the DSECT is modified to declare additional storage, then GLOB must be updated to declare additional storage.

Global Fields in Global Area 3: To address GL3 records and fields, the GLOBZ macro must be used with the REGS or REGC option. GLOBZ calls GLOBY, which in turn calls GLOBP and GLOBQ. GLOBY includes the following code:

```
@GLOBY DS 68D          DEFINE THE AREA FOR 68 SLOTS
@GLOBP DS 132D         X'84'          1056 BYTES
@GLOBQ DS 132D         X'84'          1056 BYTES
        GLOBP          NON-KEYPOINTABLE TPF GLOBAL
        GLOBQ          KEYPOINTABLE TPF GLOBAL
```

GL3 is very similar to GL1:

1. GLOBY declares storage and addresses for the GL3 directory and the global blocks.
2. The directory has exactly 68 doubleword slots declared.
3. GLOBP and GLOBQ macros address the global blocks. As in GL1, the amount of storage addressed by GLOBP and GLOBQ must not exceed the number of doublewords declared by GLOBY.
4. The amount of storage declared for each global block must exactly match the number of doublewords that will be loaded into that area (see *Number of Double Words* in “GOA List Entry” on page 304).
5. The sum of storage GLOBY declares for the directory and the global blocks must not exceed 4KB.

To define addressability to a new global field in GL3, you must modify either GLOBP or GLOBQ. As in GL1, if the amount of storage added to GLOBP or GLOBQ causes the DSECT to exceed the amount declared by GLOBY, GLOBY must be updated. For new global fields in GL3, the GLOBZ macro must also be updated. The tag for the new global field must be added to the list of subsystem user common fields in the beginning of the GLOBZ macro.

Keypointable Global Records

A global record can be keypointed if it is addressed from one of the first 48 slots of the GL1 directory or from one of the first 64 slots of the GL3 directory. The KEYUC macro should contain an entry for each of the 48 slots in GL1. The GLOUC macro should have an entry for each of the 64 slots in GL3. Global fields are considered keypointable if they reside in a keypointable global record. How to designate a global as keypointable is described under *Attribute Byte* in “GOA List Entry” on page 304.

Global Area Requirements of the Control Program

GL1 contains not only fields for application use but also fields required by the system itself. Examples are @MAXCL and @MAXBK, which are used in creating output messages. @MAXCL is a 2-byte field that contains the maximum line length for a standard CRT display, usually 64 characters. @MAXBK is a 2-byte field that contains the maximum number of characters allowed in one message block, the maximum being 342 characters per block.

A typical application field is @SWITCH, which is tested during cycling up or down to determine whether any running utility programs should be halted.

GL0BD, GL0BE, and GL0BF do not contain any fields required by the system. Usually, those fields are in GLOBP and GLOBQ.

Required System Data Resident in GLOBP

The GLOBP data macro contains the following fields used by RLCH (chain release routine) and DBR (database reorganization):

- @RLCHC and @RLCHS (used by RLCH). For more information about using these fields, see the prolog of program segment RLCH.
- @DBRKPT (used by DBR). For more information about using this field, see the database reorganization information in *TPF Database Reference* and also the prolog of program segments BDBE, BDBF, and BDBL.

Loading Globals

The loading of globals occurs in several steps:

- System Initialization Package (SIP) macros that define the size of the various global areas are coded.
- An input data set containing the super global storage allocator (super GOA), all GOAs, and all global data is coded. The global blocks included in the global data must be defined as #GLOBL fixed file records.
- The System Test Compiler (STC) is to create a pilot tape from the input deck.
- Following the general file IPL, the pilot tape is loaded onto the online modules by the online system data loader.

Note: A pilot tape load must be done for each subsystem user in the system.

- During a system IPL, the Application Core Load Program, GOGO, is activated to load the globals. GOGO retrieves the #GLOBL records from DASD and stores the global data into the global areas.

SIP for Globals

The global area you designed is created on your system by SIP macros. To find out more information on this, see *TPF System Generation*.

Creating the Input Data Set

To create TPF globals, you must first define an input data set to be used by the offline System Test Compiler (STC) to create a pilot tape (see *TPF Program Development Support Reference*). The records must be defined as #GLOBL fixed file records. An example of the STC card images used to create the global records is shown in “Sample STC Card Images for Global Block Creation” on page 314. The STC card images must include global storage allocator (GOA) records, which specify certain attributes of the other #GLOBL records to be loaded and provide information that controls the actual loading process. The DSECT that defines the fields of a GOA record is GO1GO. The contents of the pilot tape must later be transferred to DASD by the online system data loader, typically following a general file load when the system is in 1052 state.

Ordinal number 0 of #GLOBL is called the *super GOA*. The super GOA points to a chain of prime GOA records for each I-stream in a loosely coupled or tightly coupled complex. Additional ordinals can be chained to ordinal number 0 to accommodate complexes with greater than eight I-streams. See the *TPF Migration Guide: Program Update Tapes* for information about adding a SIP definition for additional super GOA ordinals. The GOA records are in #GLOBL fixed file records. They are pointed to by their ordinal number. Each GOA record (except the super GOA) contains a list of global records to be loaded. This feature makes it easy to

establish processor-unique globals, or, if desired, various levels of interdependent globals. Figure 7 is an example of the GOA data structure.

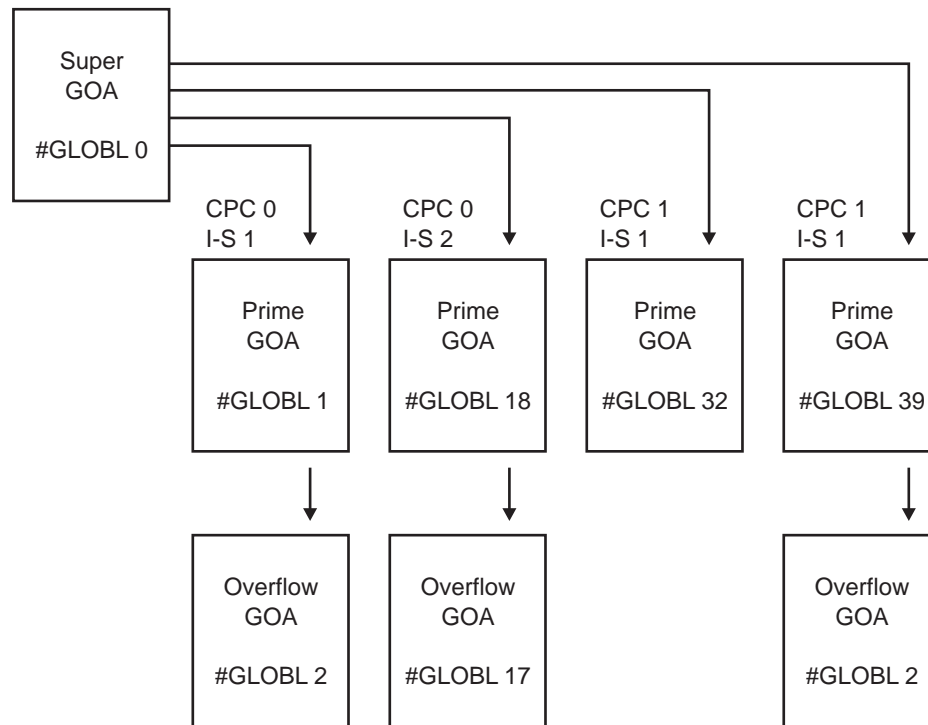


Figure 7. GOA Data Structure

All GOA records have a standard 12-byte header, except that the forward chain field contains a #GLOBL ordinal number instead of a file address. The record ID field, GO1BID, and the forward chain field, GO1FCH, must be initialized using STC. The record ID for the super GOA and all the GOA records must be C'GO'.

Super GOA

The super GOA, which is a subsystem user unique data record identifies, in the proper order each GOA, which is CPC and I-stream unique. Therefore, in a tightly coupled system, the super GOA can have some GOAs to activate and, if more than one subsystem user is present, more than one super GOA is necessary.

The super GOA has four sections:

- Section 1 is the standard header, and the super GOA uses a forward chain defined in the header to point to additional records if the definition of the data cannot be contained in the first record.
- Section 2 is the table header and contains the total number of I-streams for which a prime GOA is defined in this super GOA.
- Section 3 is a table of displacements, each pointing to an area in section 4. There is a 2-byte field (GO2DSP) in the table for each possible CPC in the system. If multiple super GOAs are used, the leftmost bit of GO2DSP must be coded as a 1 for any CPC that is not defined in that super GOA ordinal. This indicates to the GOGO program that the next chained ordinal must be read and searched for that CPC.
- Section 4 of the super GOA is also a table. Each entry in the table contains three fullword fields (GO2CID, GO2IID, and GO2IS), which list the #GLOBL ordinal

numbers for each I-stream. Given the number of the CPC (GO2CID) and the I-stream number to be loaded (GO2IID), the table provides the #GLOBL ordinal number of a prime GOA (GO2IS).

For the GOGO program to index into the tables in Sections 3 and 4 using the GO2DSP, GO2CID, GO2IID, and GO2IS field definitions, bit 2 of the 1-byte GO1CHN field in Section 2 must be set to 1 (X'40'). The use of the GO2DSP, GO2CID, GO2IID, and GO2IS field definitions is required to support 32-way loosely coupled processors.

The following example shows the STC defining two super GOA ordinals for 32 loosely coupled processors with 16 I-streams each:

```
*****
*
* TEST VERSION 'SUPERGOA' RECORD 1: 32 CPC'S, 16 I/STREAMS EACH *
*   AT PRESENT, GOA POINTERS WITHIN THE SUPERGOA ARE ALL SET TO: *
*                                     #GLOBL 1 FOR BSS *
*                                     #GLOBL 16 (X'10') FOR SSU'S *
*
*****
GO1GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)0.      THIS IS #GLOBL ZERO
GO1BID  ENT  G0.             ID = 'G0'
GO1CTL  ENT  X'00'.          N/A
GO1FCH  ENT  X'005A'.        Chained to ordinal 90
GO1NIS  ENTIT 1-1-X'0040'.    16 I/STREAMS * 4 CPCs
GO1NUM  ENTIT 1-1-X'00'.      LOAD MODE N/A
GO1CHN  ENTIT 1-1-X'40'.      GO2DSP Entries Used
*
GO2DSP  ENTIT 1-01-X'0000'.    CPC 0 INDEX
GO2DSP  ENTIT 1-02-X'0010'.    CPC 1 INDEX
GO2DSP  ENTIT 1-03-X'0020'.    CPC 2 INDEX
GO2DSP  ENTIT 1-04-X'0030'.    CPC 3 INDEX
GO2DSP  ENTIT 1-05-X'8000'.    CPC 4 NOT IN THIS RECORD
GO2DSP  ENTIT 1-06-X'8000'.    CPC 5 NOT IN THIS RECORD
GO2DSP  ENTIT 1-07-X'8000'.    CPC 6 NOT IN THIS RECORD
GO2DSP  ENTIT 1-08-X'8000'.    CPC 7 NOT IN THIS RECORD
GO2DSP  ENTIT 1-09-X'8000'.    CPC 8 NOT IN THIS RECORD
GO2DSP  ENTIT 1-10-X'8000'.    CPC 9 NOT IN THIS RECORD
GO2DSP  ENTIT 1-11-X'8000'.    CPC 10 NOT IN THIS RECORD
GO2DSP  ENTIT 1-12-X'8000'.    CPC 11 NOT IN THIS RECORD
GO2DSP  ENTIT 1-13-X'8000'.    CPC 12 NOT IN THIS RECORD
GO2DSP  ENTIT 1-14-X'8000'.    CPC 13 NOT IN THIS RECORD
GO2DSP  ENTIT 1-15-X'8000'.    CPC 14 NOT IN THIS RECORD
GO2DSP  ENTIT 1-16-X'8000'.    CPC 15 NOT IN THIS RECORD
GO2DSP  ENTIT 1-17-X'8000'.    CPC 16 NOT IN THIS RECORD
GO2DSP  ENTIT 1-18-X'8000'.    CPC 17 NOT IN THIS RECORD
GO2DSP  ENTIT 1-19-X'8000'.    CPC 18 NOT IN THIS RECORD
GO2DSP  ENTIT 1-20-X'8000'.    CPC 19 NOT IN THIS RECORD
GO2DSP  ENTIT 1-21-X'8000'.    CPC 20 NOT IN THIS RECORD
GO2DSP  ENTIT 1-22-X'8000'.    CPC 21 NOT IN THIS RECORD
GO2DSP  ENTIT 1-23-X'8000'.    CPC 22 NOT IN THIS RECORD
GO2DSP  ENTIT 1-24-X'8000'.    CPC 23 NOT IN THIS RECORD
GO2DSP  ENTIT 1-25-X'8000'.    CPC 24 NOT IN THIS RECORD
GO2DSP  ENTIT 1-26-X'8000'.    CPC 25 NOT IN THIS RECORD
GO2DSP  ENTIT 1-27-X'8000'.    CPC 26 NOT IN THIS RECORD
GO2DSP  ENTIT 1-28-X'8000'.    CPC 27 NOT IN THIS RECORD
GO2DSP  ENTIT 1-29-X'8000'.    CPC 28 NOT IN THIS RECORD
GO2DSP  ENTIT 1-30-X'8000'.    CPC 29 NOT IN THIS RECORD
GO2DSP  ENTIT 1-31-X'8000'.    CPC 30 NOT IN THIS RECORD
GO2DSP  ENTIT 1-32-X'8000'.    CPC 31 NOT IN THIS RECORD
*
GO2CID  ENTIT 1-01-X'00000000'. CPC 0
GO2IID  ENTIT 1-01-X'00000001'.    I/S 1
```

```

G02IS  ENTIT 1-01-X'00000001'.          GOA ORD NUM (#1)
*
G02CID  ENTIT 1-02-X'00000000'.  CPC 0
G02IID  ENTIT 1-02-X'00000002'.          I/S 2
G02IS  ENTIT 1-02-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-03-X'00000000'.  CPC 0
G02IID  ENTIT 1-03-X'00000003'.          I/S 3
G02IS  ENTIT 1-03-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-04-X'00000000'.  CPC 0
G02IID  ENTIT 1-04-X'00000004'.          I/S 4
G02IS  ENTIT 1-04-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-05-X'00000000'.  CPC 0
G02IID  ENTIT 1-05-X'00000005'.          I/S 5
G02IS  ENTIT 1-05-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-06-X'00000000'.  CPC 0
G02IID  ENTIT 1-06-X'00000006'.          I/S 6
G02IS  ENTIT 1-06-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-07-X'00000000'.  CPC 0
G02IID  ENTIT 1-07-X'00000007'.          I/S 7
G02IS  ENTIT 1-07-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-08-X'00000000'.  CPC 0
G02IID  ENTIT 1-08-X'00000008'.          I/S 8
G02IS  ENTIT 1-08-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-09-X'00000000'.  CPC 0
G02IID  ENTIT 1-09-X'00000009'.          I/S 9
G02IS  ENTIT 1-09-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-10-X'00000000'.  CPC 0
G02IID  ENTIT 1-10-X'0000000A'.          I/S 10
G02IS  ENTIT 1-10-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-11-X'00000000'.  CPC 0
G02IID  ENTIT 1-11-X'0000000B'.          I/S 11
G02IS  ENTIT 1-11-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-12-X'00000000'.  CPC 0
G02IID  ENTIT 1-12-X'0000000C'.          I/S 12
G02IS  ENTIT 1-12-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-13-X'00000000'.  CPC 0
G02IID  ENTIT 1-13-X'0000000D'.          I/S 13
G02IS  ENTIT 1-13-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-14-X'00000000'.  CPC 0
G02IID  ENTIT 1-14-X'0000000E'.          I/S 14
G02IS  ENTIT 1-14-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-15-X'00000000'.  CPC 0
G02IID  ENTIT 1-15-X'0000000F'.          I/S 15
G02IS  ENTIT 1-15-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-16-X'00000000'.  CPC 0
G02IID  ENTIT 1-16-X'00000010'.          I/S 16
G02IS  ENTIT 1-16-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-17-X'00000001'.  CPC 1

```

G02IID	ENTIT 1-17-X'00000001'.	I/S 1	
G02IS	ENTIT 1-17-X'00000001'.		GOA ORD NUM
*			
G02CID	ENTIT 1-18-X'00000001'.	CPC 1	
G02IID	ENTIT 1-18-X'00000002'.	I/S 2	
G02IS	ENTIT 1-18-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-19-X'00000001'.	CPC 1	
G02IID	ENTIT 1-19-X'00000003'.	I/S 3	
G02IS	ENTIT 1-19-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-20-X'00000001'.	CPC 1	
G02IID	ENTIT 1-20-X'00000004'.	I/S 4	
G02IS	ENTIT 1-20-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-21-X'00000001'.	CPC 1	
G02IID	ENTIT 1-21-X'00000005'.	I/S 5	
G02IS	ENTIT 1-21-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-22-X'00000001'.	CPC 1	
G02IID	ENTIT 1-22-X'00000006'.	I/S 6	
G02IS	ENTIT 1-22-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-23-X'00000001'.	CPC 1	
G02IID	ENTIT 1-23-X'00000007'.	I/S 7	
G02IS	ENTIT 1-23-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-24-X'00000001'.	CPC 1	
G02IID	ENTIT 1-24-X'00000008'.	I/S 8	
G02IS	ENTIT 1-24-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-25-X'00000001'.	CPC 1	
G02IID	ENTIT 1-25-X'00000009'.	I/S 9	
G02IS	ENTIT 1-25-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-26-X'00000001'.	CPC 1	
G02IID	ENTIT 1-26-X'0000000A'.	I/S 10	
G02IS	ENTIT 1-26-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-27-X'00000001'.	CPC 1	
G02IID	ENTIT 1-27-X'0000000B'.	I/S 11	
G02IS	ENTIT 1-27-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-28-X'00000001'.	CPC 1	
G02IID	ENTIT 1-28-X'0000000C'.	I/S 12	
G02IS	ENTIT 1-28-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-29-X'00000001'.	CPC 1	
G02IID	ENTIT 1-29-X'0000000D'.	I/S 13	
G02IS	ENTIT 1-29-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-30-X'00000001'.	CPC 1	
G02IID	ENTIT 1-30-X'0000000E'.	I/S 14	
G02IS	ENTIT 1-30-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-31-X'00000001'.	CPC 1	
G02IID	ENTIT 1-31-X'0000000F'.	I/S 15	
G02IS	ENTIT 1-31-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-32-X'00000001'.	CPC 1	
G02IID	ENTIT 1-32-X'00000010'.	I/S 16	
G02IS	ENTIT 1-32-X'00000010'.		GOA ORD NUM (#16)
*			

*			

```

G02CID  ENTIT 1-33-X'00000002'.  CPC 2
G02IID  ENTIT 1-33-X'00000001'.      I/S 1
G02IS   ENTIT 1-33-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-34-X'00000002'.  CPC 2
G02IID  ENTIT 1-34-X'00000002'.      I/S 2
G02IS   ENTIT 1-34-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-35-X'00000002'.  CPC 2
G02IID  ENTIT 1-35-X'00000003'.      I/S 3
G02IS   ENTIT 1-35-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-36-X'00000002'.  CPC 2
G02IID  ENTIT 1-36-X'00000004'.      I/S 4
G02IS   ENTIT 1-36-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-37-X'00000002'.  CPC 2
G02IID  ENTIT 1-37-X'00000005'.      I/S 5
G02IS   ENTIT 1-37-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'00000002'.  CPC 2
G02IID  ENTIT 1-38-X'00000006'.      I/S 6
G02IS   ENTIT 1-38-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'00000002'.  CPC 2
G02IID  ENTIT 1-39-X'00000007'.      I/S 7
G02IS   ENTIT 1-39-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'00000002'.  CPC 2
G02IID  ENTIT 1-40-X'00000008'.      I/S 8
G02IS   ENTIT 1-40-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-41-X'00000002'.  CPC 2
G02IID  ENTIT 1-41-X'00000009'.      I/S 9
G02IS   ENTIT 1-41-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'00000002'.  CPC 2
G02IID  ENTIT 1-42-X'0000000A'.      I/S 10
G02IS   ENTIT 1-42-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'00000002'.  CPC 2
G02IID  ENTIT 1-43-X'0000000B'.      I/S 11
G02IS   ENTIT 1-43-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'00000002'.  CPC 2
G02IID  ENTIT 1-44-X'0000000C'.      I/S 12
G02IS   ENTIT 1-44-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-45-X'00000002'.  CPC 2
G02IID  ENTIT 1-45-X'0000000D'.      I/S 13
G02IS   ENTIT 1-45-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'00000002'.  CPC 2
G02IID  ENTIT 1-46-X'0000000E'.      I/S 14
G02IS   ENTIT 1-46-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'00000002'.  CPC 2
G02IID  ENTIT 1-47-X'0000000F'.      I/S 15
G02IS   ENTIT 1-47-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'00000002'.  CPC 2
G02IID  ENTIT 1-48-X'00000010'.      I/S 16
G02IS   ENTIT 1-48-X'00000010'.      GOA ORD NUM (#16)
*
*****

```



```

*
G02CID ENTIT 1-49-X'00000003'. CPC 3
G02IID ENTIT 1-49-X'00000001'. I/S 1
G02IS ENTIT 1-49-X'00000001'. GOA ORD NUM
*
G02CID ENTIT 1-50-X'00000003'. CPC 3
G02IID ENTIT 1-50-X'00000002'. I/S 2
G02IS ENTIT 1-50-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-51-X'00000003'. CPC 3
G02IID ENTIT 1-51-X'00000003'. I/S 3
G02IS ENTIT 1-51-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-52-X'00000003'. CPC 3
G02IID ENTIT 1-52-X'00000004'. I/S 4
G02IS ENTIT 1-52-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-53-X'00000003'. CPC 3
G02IID ENTIT 1-53-X'00000005'. I/S 5
G02IS ENTIT 1-53-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-54-X'00000003'. CPC 3
G02IID ENTIT 1-54-X'00000006'. I/S 6
G02IS ENTIT 1-54-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-55-X'00000003'. CPC 3
G02IID ENTIT 1-55-X'00000007'. I/S 7
G02IS ENTIT 1-55-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-56-X'00000003'. CPC 3
G02IID ENTIT 1-56-X'00000008'. I/S 8
G02IS ENTIT 1-56-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-57-X'00000003'. CPC 3
G02IID ENTIT 1-57-X'00000009'. I/S 9
G02IS ENTIT 1-57-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-58-X'00000003'. CPC 3
G02IID ENTIT 1-58-X'0000000A'. I/S 10
G02IS ENTIT 1-58-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-59-X'00000003'. CPC 3
G02IID ENTIT 1-59-X'0000000B'. I/S 11
G02IS ENTIT 1-59-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-60-X'00000003'. CPC 3
G02IID ENTIT 1-60-X'0000000C'. I/S 12
G02IS ENTIT 1-60-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-61-X'00000003'. CPC 3
G02IID ENTIT 1-61-X'0000000D'. I/S 13
G02IS ENTIT 1-61-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-62-X'00000003'. CPC 3
G02IID ENTIT 1-62-X'0000000E'. I/S 14
G02IS ENTIT 1-62-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-63-X'00000003'. CPC 3
G02IID ENTIT 1-63-X'0000000F'. I/S 15
G02IS ENTIT 1-63-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-64-X'00000003'. CPC 3
G02IID ENTIT 1-64-X'00000010'. I/S 16
G02IS ENTIT 1-64-X'00000010'. GOA ORD NUM (#16)
*

```

```

GEND
*****
* This is the 2nd SUPERGOA ordinal
*
G01GO   GSTAR 1.
BSTA06  ENT   (#GLOBL)90.      THIS IS #GLOBL 90 (x'5A')
G01BID  ENT   GO.              ID = 'GO'
G01CTL  ENT   X'00'.           N/A
G01FCH  ENT   X'005B'.         Chained to ordinal 91
G01NIS  ENTIT 1-1-X'0040'.     16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.       LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.       G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'.     CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'.     CPC 1 NOT IN THIS RECORD
G02DSP  ENTIT 1-03-X'8000'.     CPC 2 NOT IN THIS RECORD
G02DSP  ENTIT 1-04-X'8000'.     CPC 3 NOT IN THIS RECORD
G02DSP  ENTIT 1-05-X'0000'.     CPC 4 INDEX
G02DSP  ENTIT 1-06-X'0010'.     CPC 5 INDEX
G02DSP  ENTIT 1-07-X'0020'.     CPC 6 INDEX
G02DSP  ENTIT 1-08-X'0030'.     CPC 7 INDEX
G02DSP  ENTIT 1-09-X'8000'.     CPC 8 NOT IN THIS RECORD
G02DSP  ENTIT 1-10-X'8000'.     CPC 9 NOT IN THIS RECORD
G02DSP  ENTIT 1-11-X'8000'.     CPC 10 NOT IN THIS RECORD
G02DSP  ENTIT 1-12-X'8000'.     CPC 11 NOT IN THIS RECORD
G02DSP  ENTIT 1-13-X'8000'.     CPC 12 NOT IN THIS RECORD
G02DSP  ENTIT 1-14-X'8000'.     CPC 13 NOT IN THIS RECORD
G02DSP  ENTIT 1-15-X'8000'.     CPC 14 NOT IN THIS RECORD
G02DSP  ENTIT 1-16-X'8000'.     CPC 15 NOT IN THIS RECORD
G02DSP  ENTIT 1-17-X'8000'.     CPC 16 NOT IN THIS RECORD
G02DSP  ENTIT 1-18-X'8000'.     CPC 17 NOT IN THIS RECORD
G02DSP  ENTIT 1-19-X'8000'.     CPC 18 NOT IN THIS RECORD
G02DSP  ENTIT 1-20-X'8000'.     CPC 19 NOT IN THIS RECORD
G02DSP  ENTIT 1-21-X'8000'.     CPC 20 NOT IN THIS RECORD
G02DSP  ENTIT 1-22-X'8000'.     CPC 21 NOT IN THIS RECORD
G02DSP  ENTIT 1-23-X'8000'.     CPC 22 NOT IN THIS RECORD
G02DSP  ENTIT 1-24-X'8000'.     CPC 23 NOT IN THIS RECORD
G02DSP  ENTIT 1-25-X'8000'.     CPC 24 NOT IN THIS RECORD
G02DSP  ENTIT 1-26-X'8000'.     CPC 25 NOT IN THIS RECORD
G02DSP  ENTIT 1-27-X'8000'.     CPC 26 NOT IN THIS RECORD
G02DSP  ENTIT 1-28-X'8000'.     CPC 27 NOT IN THIS RECORD
G02DSP  ENTIT 1-29-X'8000'.     CPC 28 NOT IN THIS RECORD
G02DSP  ENTIT 1-30-X'8000'.     CPC 29 NOT IN THIS RECORD
G02DSP  ENTIT 1-31-X'8000'.     CPC 30 NOT IN THIS RECORD
G02DSP  ENTIT 1-32-X'8000'.     CPC 31 NOT IN THIS RECORD
*
*
G02CID  ENTIT 1-01-X'00000004'. CPC 4
G02IID  ENTIT 1-01-X'00000001'.   I/S 1
G02IS   ENTIT 1-01-X'00000001'.   GOA ORD NUM
*
G02CID  ENTIT 1-02-X'00000004'. CPC 4
G02IID  ENTIT 1-02-X'00000002'.   I/S 2
G02IS   ENTIT 1-02-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-03-X'00000004'. CPC 4
G02IID  ENTIT 1-03-X'00000003'.   I/S 3
G02IS   ENTIT 1-03-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-04-X'00000004'. CPC 4
G02IID  ENTIT 1-04-X'00000004'.   I/S 4
G02IS   ENTIT 1-04-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-05-X'00000004'. CPC 4
G02IID  ENTIT 1-05-X'00000005'.   I/S 5
G02IS   ENTIT 1-05-X'00000010'.   GOA ORD NUM (#16)
*

```

G02CID	ENTIT 1-06-X'00000004'.	CPC 4	
G02IID	ENTIT 1-06-X'00000006'.		I/S 6
G02IS	ENTIT 1-06-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-07-X'00000004'.	CPC 4	
G02IID	ENTIT 1-07-X'00000007'.		I/S 7
G02IS	ENTIT 1-07-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-08-X'00000004'.	CPC 4	
G02IID	ENTIT 1-08-X'00000008'.		I/S 8
G02IS	ENTIT 1-08-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-09-X'00000004'.	CPC 4	
G02IID	ENTIT 1-09-X'00000009'.		I/S 9
G02IS	ENTIT 1-09-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-10-X'00000004'.	CPC 4	
G02IID	ENTIT 1-10-X'0000000A'.		I/S 10
G02IS	ENTIT 1-10-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-11-X'00000004'.	CPC 4	
G02IID	ENTIT 1-11-X'0000000B'.		I/S 11
G02IS	ENTIT 1-11-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-12-X'00000004'.	CPC 4	
G02IID	ENTIT 1-12-X'0000000C'.		I/S 12
G02IS	ENTIT 1-12-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-13-X'00000004'.	CPC 4	
G02IID	ENTIT 1-13-X'0000000D'.		I/S 13
G02IS	ENTIT 1-13-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-14-X'00000004'.	CPC 4	
G02IID	ENTIT 1-14-X'0000000E'.		I/S 14
G02IS	ENTIT 1-14-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-15-X'00000004'.	CPC 4	
G02IID	ENTIT 1-15-X'0000000F'.		I/S 15
G02IS	ENTIT 1-15-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-16-X'00000004'.	CPC 4	
G02IID	ENTIT 1-16-X'00000010'.		I/S 16
G02IS	ENTIT 1-16-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-17-X'00000005'.	CPC 5	
G02IID	ENTIT 1-17-X'00000001'.		I/S 1
G02IS	ENTIT 1-17-X'00000001'.		GOA ORD NUM
*			
G02CID	ENTIT 1-18-X'00000005'.	CPC 5	
G02IID	ENTIT 1-18-X'00000002'.		I/S 2
G02IS	ENTIT 1-18-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-19-X'00000005'.	CPC 5	
G02IID	ENTIT 1-19-X'00000003'.		I/S 3
G02IS	ENTIT 1-19-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-20-X'00000005'.	CPC 5	
G02IID	ENTIT 1-20-X'00000004'.		I/S 4
G02IS	ENTIT 1-20-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-21-X'00000005'.	CPC 5	
G02IID	ENTIT 1-21-X'00000005'.		I/S 5
G02IS	ENTIT 1-21-X'00000010'.		GOA ORD NUM (#16)

```

*
G02CID ENTIT 1-22-X'00000005'. CPC 5
G02IID ENTIT 1-22-X'00000006'. I/S 6
G02IS ENTIT 1-22-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-23-X'00000005'. CPC 5
G02IID ENTIT 1-23-X'00000007'. I/S 7
G02IS ENTIT 1-23-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-24-X'00000005'. CPC 5
G02IID ENTIT 1-24-X'00000008'. I/S 8
G02IS ENTIT 1-24-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-25-X'00000005'. CPC 5
G02IID ENTIT 1-25-X'00000009'. I/S 9
G02IS ENTIT 1-25-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-26-X'00000005'. CPC 5
G02IID ENTIT 1-26-X'0000000A'. I/S 10
G02IS ENTIT 1-26-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-27-X'00000005'. CPC 5
G02IID ENTIT 1-27-X'0000000B'. I/S 11
G02IS ENTIT 1-27-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-28-X'00000005'. CPC 5
G02IID ENTIT 1-28-X'0000000C'. I/S 12
G02IS ENTIT 1-28-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-29-X'00000005'. CPC 5
G02IID ENTIT 1-29-X'0000000D'. I/S 13
G02IS ENTIT 1-29-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-30-X'00000005'. CPC 5
G02IID ENTIT 1-30-X'0000000E'. I/S 14
G02IS ENTIT 1-30-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-31-X'00000005'. CPC 5
G02IID ENTIT 1-31-X'0000000F'. I/S 15
G02IS ENTIT 1-31-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-32-X'00000005'. CPC 5
G02IID ENTIT 1-32-X'00000010'. I/S 16
G02IS ENTIT 1-32-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-33-X'00000006'. CPC 6
G02IID ENTIT 1-33-X'00000001'. I/S 1
G02IS ENTIT 1-33-X'00000001'. GOA ORD NUM
*
G02CID ENTIT 1-34-X'00000006'. CPC 6
G02IID ENTIT 1-34-X'00000002'. I/S 2
G02IS ENTIT 1-34-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-35-X'00000006'. CPC 6
G02IID ENTIT 1-35-X'00000003'. I/S 3
G02IS ENTIT 1-35-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-36-X'00000006'. CPC 6
G02IID ENTIT 1-36-X'00000004'. I/S 4
G02IS ENTIT 1-36-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-37-X'00000006'. CPC 6
G02IID ENTIT 1-37-X'00000005'. I/S 5

```

```

G02IS  ENTIT 1-37-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'00000006'.    CPC 6
G02IID  ENTIT 1-38-X'00000006'.    I/S 6
G02IS   ENTIT 1-38-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'00000006'.    CPC 6
G02IID  ENTIT 1-39-X'00000007'.    I/S 7
G02IS   ENTIT 1-39-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'00000006'.    CPC 6
G02IID  ENTIT 1-40-X'00000008'.    I/S 8
G02IS   ENTIT 1-40-X'00000010'.    GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-41-X'00000006'.    CPC 6
G02IID  ENTIT 1-41-X'00000009'.    I/S 9
G02IS   ENTIT 1-41-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'00000006'.    CPC 6
G02IID  ENTIT 1-42-X'0000000A'.    I/S 10
G02IS   ENTIT 1-42-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'00000006'.    CPC 6
G02IID  ENTIT 1-43-X'0000000B'.    I/S 11
G02IS   ENTIT 1-43-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'00000006'.    CPC 6
G02IID  ENTIT 1-44-X'0000000C'.    I/S 12
G02IS   ENTIT 1-44-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-45-X'00000006'.    CPC 6
G02IID  ENTIT 1-45-X'0000000D'.    I/S 13
G02IS   ENTIT 1-45-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'00000006'.    CPC 6
G02IID  ENTIT 1-46-X'0000000E'.    I/S 14
G02IS   ENTIT 1-46-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'00000006'.    CPC 6
G02IID  ENTIT 1-47-X'0000000F'.    I/S 15
G02IS   ENTIT 1-47-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'00000006'.    CPC 6
G02IID  ENTIT 1-48-X'00000010'.    I/S 16
G02IS   ENTIT 1-48-X'00000010'.    GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-49-X'00000007'.    CPC 7
G02IID  ENTIT 1-49-X'00000001'.    I/S 1
G02IS   ENTIT 1-49-X'00000001'.    GOA ORD NUM
*
G02CID  ENTIT 1-50-X'00000007'.    CPC 7
G02IID  ENTIT 1-50-X'00000002'.    I/S 2
G02IS   ENTIT 1-50-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'00000007'.    CPC 7
G02IID  ENTIT 1-51-X'00000003'.    I/S 3
G02IS   ENTIT 1-51-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'00000007'.    CPC 7
G02IID  ENTIT 1-52-X'00000004'.    I/S 4
G02IS   ENTIT 1-52-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-53-X'00000007'.    CPC 7

```

```

G02IID  ENTIT 1-53-X'00000005'.      I/S 5
G02IS   ENTIT 1-53-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'00000007'.      CPC 7
G02IID  ENTIT 1-54-X'00000006'.      I/S 6
G02IS   ENTIT 1-54-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'00000007'.      CPC 7
G02IID  ENTIT 1-55-X'00000007'.      I/S 7
G02IS   ENTIT 1-55-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'00000007'.      CPC 7
G02IID  ENTIT 1-56-X'00000008'.      I/S 8
G02IS   ENTIT 1-56-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-57-X'00000007'.      CPC 7
G02IID  ENTIT 1-57-X'00000009'.      I/S 9
G02IS   ENTIT 1-57-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'00000007'.      CPC 7
G02IID  ENTIT 1-58-X'0000000A'.      I/S 10
G02IS   ENTIT 1-58-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'00000007'.      CPC 7
G02IID  ENTIT 1-59-X'0000000B'.      I/S 11
G02IS   ENTIT 1-59-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'00000007'.      CPC 7
G02IID  ENTIT 1-60-X'0000000C'.      I/S 12
G02IS   ENTIT 1-60-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'00000007'.      CPC 7
G02IID  ENTIT 1-61-X'0000000D'.      I/S 13
G02IS   ENTIT 1-61-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'00000007'.      CPC 7
G02IID  ENTIT 1-62-X'0000000E'.      I/S 14
G02IS   ENTIT 1-62-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'00000007'.      CPC 7
G02IID  ENTIT 1-63-X'0000000F'.      I/S 15
G02IS   ENTIT 1-63-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'00000007'.      CPC 7
G02IID  ENTIT 1-64-X'00000010'.      I/S 16
G02IS   ENTIT 1-64-X'00000010'.      GOA ORD NUM (#16)
*
*
*****
GEND
*****
* This is the 3rd SUPERGOA ordinal
*
G01GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)91.      THIS IS #GLOBL 91 (x'5B')
G01BID  ENT  GO.              ID = 'GO'
G01CTL  ENT  X'00'.           N/A
G01FCH  ENT  X'005C'.         Chained to #GLOBL 92
G01NIS  ENTIT 1-1-X'0040'.    16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.      LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.      G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'.    CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'.    CPC 1 NOT IN THIS RECORD
G02DSP  ENTIT 1-03-X'8000'.    CPC 2 NOT IN THIS RECORD

```

G02DSP	ENTIT 1-04-X'8000'.	CPC 3 NOT IN THIS RECORD
G02DSP	ENTIT 1-05-X'8000'.	CPC 4 NOT IN THIS RECORD
G02DSP	ENTIT 1-06-X'8000'.	CPC 5 NOT IN THIS RECORD
G02DSP	ENTIT 1-07-X'8000'.	CPC 6 NOT IN THIS RECORD
G02DSP	ENTIT 1-08-X'8000'.	CPC 7 NOT IN THIS RECORD
G02DSP	ENTIT 1-09-X'0000'.	CPC 8 INDEX
G02DSP	ENTIT 1-10-X'0010'.	CPC 9 INDEX
G02DSP	ENTIT 1-11-X'0020'.	CPC 10 INDEX
G02DSP	ENTIT 1-12-X'0030'.	CPC 11 INDEX
G02DSP	ENTIT 1-13-X'8000'.	CPC 12 NOT IN THIS RECORD
G02DSP	ENTIT 1-14-X'8000'.	CPC 13 NOT IN THIS RECORD
G02DSP	ENTIT 1-15-X'8000'.	CPC 14 NOT IN THIS RECORD
G02DSP	ENTIT 1-16-X'8000'.	CPC 15 NOT IN THIS RECORD
G02DSP	ENTIT 1-17-X'8000'.	CPC 16 NOT IN THIS RECORD
G02DSP	ENTIT 1-18-X'8000'.	CPC 17 NOT IN THIS RECORD
G02DSP	ENTIT 1-19-X'8000'.	CPC 18 NOT IN THIS RECORD
G02DSP	ENTIT 1-20-X'8000'.	CPC 19 NOT IN THIS RECORD
G02DSP	ENTIT 1-21-X'8000'.	CPC 20 NOT IN THIS RECORD
G02DSP	ENTIT 1-22-X'8000'.	CPC 21 NOT IN THIS RECORD
G02DSP	ENTIT 1-23-X'8000'.	CPC 22 NOT IN THIS RECORD
G02DSP	ENTIT 1-24-X'8000'.	CPC 23 NOT IN THIS RECORD
G02DSP	ENTIT 1-25-X'8000'.	CPC 24 NOT IN THIS RECORD
G02DSP	ENTIT 1-26-X'8000'.	CPC 25 NOT IN THIS RECORD
G02DSP	ENTIT 1-27-X'8000'.	CPC 26 NOT IN THIS RECORD
G02DSP	ENTIT 1-28-X'8000'.	CPC 27 NOT IN THIS RECORD
G02DSP	ENTIT 1-29-X'8000'.	CPC 28 NOT IN THIS RECORD
G02DSP	ENTIT 1-30-X'8000'.	CPC 29 NOT IN THIS RECORD
G02DSP	ENTIT 1-31-X'8000'.	CPC 30 NOT IN THIS RECORD
G02DSP	ENTIT 1-32-X'8000'.	CPC 31 NOT IN THIS RECORD
*		
G02CID	ENTIT 1-01-X'00000008'.	CPC 8
G02IID	ENTIT 1-01-X'00000001'.	I/S 1
G02IS	ENTIT 1-01-X'00000001'.	GOA ORD NUM (#1)
*		
G02CID	ENTIT 1-02-X'00000008'.	CPC 8
G02IID	ENTIT 1-02-X'00000002'.	I/S 2
G02IS	ENTIT 1-02-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-03-X'00000008'.	CPC 8
G02IID	ENTIT 1-03-X'00000003'.	I/S 3
G02IS	ENTIT 1-03-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-04-X'00000008'.	CPC 8
G02IID	ENTIT 1-04-X'00000004'.	I/S 4
G02IS	ENTIT 1-04-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-05-X'00000008'.	CPC 8
G02IID	ENTIT 1-05-X'00000005'.	I/S 5
G02IS	ENTIT 1-05-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-06-X'00000008'.	CPC 8
G02IID	ENTIT 1-06-X'00000006'.	I/S 6
G02IS	ENTIT 1-06-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-07-X'00000008'.	CPC 8
G02IID	ENTIT 1-07-X'00000007'.	I/S 7
G02IS	ENTIT 1-07-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-08-X'00000008'.	CPC 8
G02IID	ENTIT 1-08-X'00000008'.	I/S 8
G02IS	ENTIT 1-08-X'00000010'.	GOA ORD NUM (#16)
*		

*		
G02CID	ENTIT 1-09-X'00000008'.	CPC 8
G02IID	ENTIT 1-09-X'00000009'.	I/S 9
G02IS	ENTIT 1-09-X'00000010'.	GOA ORD NUM (#16)

```

*
G02CID  ENTIT 1-10-X'00000008'.   CPC 8
G02IID  ENTIT 1-10-X'0000000A'.   I/S 10
G02IS   ENTIT 1-10-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-11-X'00000008'.   CPC 8
G02IID  ENTIT 1-11-X'0000000B'.   I/S 11
G02IS   ENTIT 1-11-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-12-X'00000008'.   CPC 8
G02IID  ENTIT 1-12-X'0000000C'.   I/S 12
G02IS   ENTIT 1-12-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-13-X'00000008'.   CPC 8
G02IID  ENTIT 1-13-X'0000000D'.   I/S 13
G02IS   ENTIT 1-13-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-14-X'00000008'.   CPC 8
G02IID  ENTIT 1-14-X'0000000E'.   I/S 14
G02IS   ENTIT 1-14-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-15-X'00000008'.   CPC 8
G02IID  ENTIT 1-15-X'0000000F'.   I/S 15
G02IS   ENTIT 1-15-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-16-X'00000008'.   CPC 8
G02IID  ENTIT 1-16-X'00000010'.   I/S 16
G02IS   ENTIT 1-16-X'00000010'.   GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-17-X'00000009'.   CPC 9
G02IID  ENTIT 1-17-X'00000001'.   I/S 1
G02IS   ENTIT 1-17-X'00000001'.   GOA ORD NUM
*
G02CID  ENTIT 1-18-X'00000009'.   CPC 9
G02IID  ENTIT 1-18-X'00000002'.   I/S 2
G02IS   ENTIT 1-18-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-19-X'00000009'.   CPC 9
G02IID  ENTIT 1-19-X'00000003'.   I/S 3
G02IS   ENTIT 1-19-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-20-X'00000009'.   CPC 9
G02IID  ENTIT 1-20-X'00000004'.   I/S 4
G02IS   ENTIT 1-20-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-21-X'00000009'.   CPC 9
G02IID  ENTIT 1-21-X'00000005'.   I/S 5
G02IS   ENTIT 1-21-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-22-X'00000009'.   CPC 9
G02IID  ENTIT 1-22-X'00000006'.   I/S 6
G02IS   ENTIT 1-22-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-23-X'00000009'.   CPC 9
G02IID  ENTIT 1-23-X'00000007'.   I/S 7
G02IS   ENTIT 1-23-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-24-X'00000009'.   CPC 9
G02IID  ENTIT 1-24-X'00000008'.   I/S 8
G02IS   ENTIT 1-24-X'00000010'.   GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-25-X'00000009'.   CPC 9
G02IID  ENTIT 1-25-X'00000009'.   I/S 9

```


G02IS	ENTIT 1-25-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-26-X'00000009'.	CPC 9	
G02IID	ENTIT 1-26-X'0000000A'.	I/S 10	
G02IS	ENTIT 1-26-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-27-X'00000009'.	CPC 9	
G02IID	ENTIT 1-27-X'0000000B'.	I/S 11	
G02IS	ENTIT 1-27-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-28-X'00000009'.	CPC 9	
G02IID	ENTIT 1-28-X'0000000C'.	I/S 12	
G02IS	ENTIT 1-28-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-29-X'00000009'.	CPC 9	
G02IID	ENTIT 1-29-X'0000000D'.	I/S 13	
G02IS	ENTIT 1-29-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-30-X'00000009'.	CPC 9	
G02IID	ENTIT 1-30-X'0000000E'.	I/S 14	
G02IS	ENTIT 1-30-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-31-X'00000009'.	CPC 9	
G02IID	ENTIT 1-31-X'0000000F'.	I/S 15	
G02IS	ENTIT 1-31-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-32-X'00000009'.	CPC 9	
G02IID	ENTIT 1-32-X'00000010'.	I/S 16	
G02IS	ENTIT 1-32-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-33-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-33-X'00000001'.	I/S 1	
G02IS	ENTIT 1-33-X'00000001'.		GOA ORD NUM
*			
G02CID	ENTIT 1-34-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-34-X'00000002'.	I/S 2	
G02IS	ENTIT 1-34-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-35-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-35-X'00000003'.	I/S 3	
G02IS	ENTIT 1-35-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-36-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-36-X'00000004'.	I/S 4	
G02IS	ENTIT 1-36-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-37-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-37-X'00000005'.	I/S 5	
G02IS	ENTIT 1-37-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-38-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-38-X'00000006'.	I/S 6	
G02IS	ENTIT 1-38-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-39-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-39-X'00000007'.	I/S 7	
G02IS	ENTIT 1-39-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-40-X'0000000A'.	CPC 10	
G02IID	ENTIT 1-40-X'00000008'.	I/S 8	
G02IS	ENTIT 1-40-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-41-X'0000000A'.	CPC 10	

```

G02IID  ENTIT 1-41-X'00000009'.      I/S 9
G02IS   ENTIT 1-41-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'0000000A'.      CPC 10
G02IID  ENTIT 1-42-X'0000000A'.      I/S 10
G02IS   ENTIT 1-42-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'0000000A'.      CPC 10
G02IID  ENTIT 1-43-X'0000000B'.      I/S 11
G02IS   ENTIT 1-43-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'0000000A'.      CPC 10
G02IID  ENTIT 1-44-X'0000000C'.      I/S 12
G02IS   ENTIT 1-44-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-45-X'0000000A'.      CPC 10
G02IID  ENTIT 1-45-X'0000000D'.      I/S 13
G02IS   ENTIT 1-45-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'0000000A'.      CPC 10
G02IID  ENTIT 1-46-X'0000000E'.      I/S 14
G02IS   ENTIT 1-46-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'0000000A'.      CPC 10
G02IID  ENTIT 1-47-X'0000000F'.      I/S 15
G02IS   ENTIT 1-47-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'0000000A'.      CPC 10
G02IID  ENTIT 1-48-X'00000010'.      I/S 16
G02IS   ENTIT 1-48-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-49-X'0000000B'.      CPC 11
G02IID  ENTIT 1-49-X'00000001'.      I/S 1
G02IS   ENTIT 1-49-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-50-X'0000000B'.      CPC 11
G02IID  ENTIT 1-50-X'00000002'.      I/S 2
G02IS   ENTIT 1-50-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'0000000B'.      CPC 11
G02IID  ENTIT 1-51-X'00000003'.      I/S 3
G02IS   ENTIT 1-51-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'0000000B'.      CPC 11
G02IID  ENTIT 1-52-X'00000004'.      I/S 4
G02IS   ENTIT 1-52-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-53-X'0000000B'.      CPC 11
G02IID  ENTIT 1-53-X'00000005'.      I/S 5
G02IS   ENTIT 1-53-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'0000000B'.      CPC 11
G02IID  ENTIT 1-54-X'00000006'.      I/S 6
G02IS   ENTIT 1-54-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'0000000B'.      CPC 11
G02IID  ENTIT 1-55-X'00000007'.      I/S 7
G02IS   ENTIT 1-55-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'0000000B'.      CPC 11
G02IID  ENTIT 1-56-X'00000008'.      I/S 8
G02IS   ENTIT 1-56-X'00000010'.      GOA ORD NUM (#16)
*
*****
*

```

```

G02CID  ENTIT 1-57-X'0000000B'.  CPC 11
G02IID  ENTIT 1-57-X'00000009'.    I/S 9
G02IS   ENTIT 1-57-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'0000000B'.  CPC 11
G02IID  ENTIT 1-58-X'0000000A'.    I/S 10
G02IS   ENTIT 1-58-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'0000000B'.  CPC 11
G02IID  ENTIT 1-59-X'0000000B'.    I/S 11
G02IS   ENTIT 1-59-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'0000000B'.  CPC 11
G02IID  ENTIT 1-60-X'0000000C'.    I/S 12
G02IS   ENTIT 1-60-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'0000000B'.  CPC 11
G02IID  ENTIT 1-61-X'0000000D'.    I/S 13
G02IS   ENTIT 1-61-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'0000000B'.  CPC 11
G02IID  ENTIT 1-62-X'0000000E'.    I/S 14
G02IS   ENTIT 1-62-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'0000000B'.  CPC 11
G02IID  ENTIT 1-63-X'0000000F'.    I/S 15
G02IS   ENTIT 1-63-X'00000010'.    GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'0000000B'.  CPC 11
G02IID  ENTIT 1-64-X'00000010'.    I/S 16
G02IS   ENTIT 1-64-X'00000010'.    GOA ORD NUM (#16)
*

```

GEND

* This is the 4th SUPERGOA ordinal

*

```

G01GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)92.  THIS IS #GLOBL 92 (x'5C')
G01BID  ENT  G0.         ID = 'G0'
G01CTL  ENT  X'00'.      N/A
G01FCH  ENT  X'005D'.    Chained to ordinal 93
G01NIS  ENTIT 1-1-X'0040'. 16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.  LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.  G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'. CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'. CPC 1 NOT IN THIS RECORD
G02DSP  ENTIT 1-03-X'8000'. CPC 2 NOT IN THIS RECORD
G02DSP  ENTIT 1-04-X'8000'. CPC 3 NOT IN THIS RECORD
G02DSP  ENTIT 1-05-X'8000'. CPC 4 NOT IN THIS RECORD
G02DSP  ENTIT 1-06-X'8000'. CPC 5 NOT IN THIS RECORD
G02DSP  ENTIT 1-07-X'8000'. CPC 6 NOT IN THIS RECORD
G02DSP  ENTIT 1-08-X'8000'. CPC 7 NOT IN THIS RECORD
G02DSP  ENTIT 1-09-X'8000'. CPC 8 NOT IN THIS RECORD
G02DSP  ENTIT 1-10-X'8000'. CPC 9 NOT IN THIS RECORD
G02DSP  ENTIT 1-11-X'8000'. CPC 10 NOT IN THIS RECORD
G02DSP  ENTIT 1-12-X'8000'. CPC 11 NOT IN THIS RECORD
G02DSP  ENTIT 1-13-X'0000'. CPC 12 - 16 I STREAMS
G02DSP  ENTIT 1-14-X'0010'. CPC 13 - 16 I STREAMS
G02DSP  ENTIT 1-15-X'0020'. CPC 14 - 16 I STREAMS
G02DSP  ENTIT 1-16-X'0030'. CPC 15 - 16 I STREAMS
G02DSP  ENTIT 1-17-X'8000'. CPC 16 NOT IN THIS RECORD
G02DSP  ENTIT 1-18-X'8000'. CPC 17 NOT IN THIS RECORD
G02DSP  ENTIT 1-19-X'8000'. CPC 18 NOT IN THIS RECORD
G02DSP  ENTIT 1-20-X'8000'. CPC 19 NOT IN THIS RECORD
G02DSP  ENTIT 1-21-X'8000'. CPC 20 NOT IN THIS RECORD

```

G02DSP	ENTIT 1-22-X'8000'.	CPC 21 NOT IN THIS RECORD
G02DSP	ENTIT 1-23-X'8000'.	CPC 22 NOT IN THIS RECORD
G02DSP	ENTIT 1-24-X'8000'.	CPC 23 NOT IN THIS RECORD
G02DSP	ENTIT 1-25-X'8000'.	CPC 24 NOT IN THIS RECORD
G02DSP	ENTIT 1-26-X'8000'.	CPC 25 NOT IN THIS RECORD
G02DSP	ENTIT 1-27-X'8000'.	CPC 26 NOT IN THIS RECORD
G02DSP	ENTIT 1-28-X'8000'.	CPC 27 NOT IN THIS RECORD
G02DSP	ENTIT 1-29-X'8000'.	CPC 28 NOT IN THIS RECORD
G02DSP	ENTIT 1-30-X'8000'.	CPC 29 NOT IN THIS RECORD
G02DSP	ENTIT 1-31-X'8000'.	CPC 30 NOT IN THIS RECORD
G02DSP	ENTIT 1-32-X'8000'.	CPC 31 NOT IN THIS RECORD
*		
G02CID	ENTIT 1-01-X'0000000C'.	CPC 12
G02IID	ENTIT 1-01-X'00000001'.	I/S 1
G02IS	ENTIT 1-01-X'00000001'.	GOA ORD NUM
*		
G02CID	ENTIT 1-02-X'0000000C'.	CPC 12
G02IID	ENTIT 1-02-X'00000002'.	I/S 2
G02IS	ENTIT 1-02-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-03-X'0000000C'.	CPC 12
G02IID	ENTIT 1-03-X'00000003'.	I/S 3
G02IS	ENTIT 1-03-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-04-X'0000000C'.	CPC 12
G02IID	ENTIT 1-04-X'00000004'.	I/S 4
G02IS	ENTIT 1-04-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-05-X'0000000C'.	CPC 12
G02IID	ENTIT 1-05-X'00000005'.	I/S 5
G02IS	ENTIT 1-05-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-06-X'0000000C'.	CPC 12
G02IID	ENTIT 1-06-X'00000006'.	I/S 6
G02IS	ENTIT 1-06-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-07-X'0000000C'.	CPC 12
G02IID	ENTIT 1-07-X'00000007'.	I/S 7
G02IS	ENTIT 1-07-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-08-X'0000000C'.	CPC 12
G02IID	ENTIT 1-08-X'00000008'.	I/S 8
G02IS	ENTIT 1-08-X'00000010'.	GOA ORD NUM (#16)
*		

*		
G02CID	ENTIT 1-09-X'0000000C'.	CPC 12
G02IID	ENTIT 1-09-X'00000009'.	I/S 9
G02IS	ENTIT 1-09-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-10-X'0000000C'.	CPC 12
G02IID	ENTIT 1-10-X'0000000A'.	I/S 10
G02IS	ENTIT 1-10-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-11-X'0000000C'.	CPC 12
G02IID	ENTIT 1-11-X'0000000B'.	I/S 11
G02IS	ENTIT 1-11-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-12-X'0000000C'.	CPC 12
G02IID	ENTIT 1-12-X'0000000C'.	I/S 12
G02IS	ENTIT 1-12-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-13-X'0000000C'.	CPC 12
G02IID	ENTIT 1-13-X'0000000D'.	I/S 13
G02IS	ENTIT 1-13-X'00000010'.	GOA ORD NUM (#16)
*		

```

G02CID  ENTIT 1-14-X'0000000C'.  CPC 12
G02IID  ENTIT 1-14-X'0000000E'.      I/S 14
G02IS   ENTIT 1-14-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-15-X'0000000C'.  CPC 12
G02IID  ENTIT 1-15-X'0000000F'.      I/S 15
G02IS   ENTIT 1-15-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-16-X'0000000C'.  CPC 12
G02IID  ENTIT 1-16-X'00000010'.      I/S 16
G02IS   ENTIT 1-16-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-17-X'0000000D'.  CPC 13
G02IID  ENTIT 1-17-X'00000001'.      I/S 1
G02IS   ENTIT 1-17-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-18-X'0000000D'.  CPC 13
G02IID  ENTIT 1-18-X'00000002'.      I/S 2
G02IS   ENTIT 1-18-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-19-X'0000000D'.  CPC 13
G02IID  ENTIT 1-19-X'00000003'.      I/S 3
G02IS   ENTIT 1-19-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-20-X'0000000D'.  CPC 13
G02IID  ENTIT 1-20-X'00000004'.      I/S 4
G02IS   ENTIT 1-20-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-21-X'0000000D'.  CPC 13
G02IID  ENTIT 1-21-X'00000005'.      I/S 5
G02IS   ENTIT 1-21-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-22-X'0000000D'.  CPC 13
G02IID  ENTIT 1-22-X'00000006'.      I/S 6
G02IS   ENTIT 1-22-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-23-X'0000000D'.  CPC 13
G02IID  ENTIT 1-23-X'00000007'.      I/S 7
G02IS   ENTIT 1-23-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-24-X'0000000D'.  CPC 13
G02IID  ENTIT 1-24-X'00000008'.      I/S 8
G02IS   ENTIT 1-24-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-25-X'0000000D'.  CPC 13
G02IID  ENTIT 1-25-X'00000009'.      I/S 9
G02IS   ENTIT 1-25-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-26-X'0000000D'.  CPC 13
G02IID  ENTIT 1-26-X'0000000A'.      I/S 10
G02IS   ENTIT 1-26-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-27-X'0000000D'.  CPC 13
G02IID  ENTIT 1-27-X'0000000B'.      I/S 11
G02IS   ENTIT 1-27-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-28-X'0000000D'.  CPC 13
G02IID  ENTIT 1-28-X'0000000C'.      I/S 12
G02IS   ENTIT 1-28-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-29-X'0000000D'.  CPC 13
G02IID  ENTIT 1-29-X'0000000D'.      I/S 13
G02IS   ENTIT 1-29-X'00000010'.      GOA ORD NUM (#16)

```

```

*
G02CID  ENTIT 1-30-X'0000000D'.  CPC 13
G02IID  ENTIT 1-30-X'0000000E'.      I/S 14
G02IS   ENTIT 1-30-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-31-X'0000000D'.  CPC 13
G02IID  ENTIT 1-31-X'0000000F'.      I/S 15
G02IS   ENTIT 1-31-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-32-X'0000000D'.  CPC 13
G02IID  ENTIT 1-32-X'00000010'.      I/S 16
G02IS   ENTIT 1-32-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-33-X'0000000E'.  CPC 14
G02IID  ENTIT 1-33-X'00000001'.      I/S 1
G02IS   ENTIT 1-33-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-34-X'0000000E'.  CPC 14
G02IID  ENTIT 1-34-X'00000002'.      I/S 2
G02IS   ENTIT 1-34-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-35-X'0000000E'.  CPC 14
G02IID  ENTIT 1-35-X'00000003'.      I/S 3
G02IS   ENTIT 1-35-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-36-X'0000000E'.  CPC 14
G02IID  ENTIT 1-36-X'00000004'.      I/S 4
G02IS   ENTIT 1-36-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-37-X'0000000E'.  CPC 14
G02IID  ENTIT 1-37-X'00000005'.      I/S 5
G02IS   ENTIT 1-37-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'0000000E'.  CPC 14
G02IID  ENTIT 1-38-X'00000006'.      I/S 6
G02IS   ENTIT 1-38-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'0000000E'.  CPC 14
G02IID  ENTIT 1-39-X'00000007'.      I/S 7
G02IS   ENTIT 1-39-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'0000000E'.  CPC 14
G02IID  ENTIT 1-40-X'00000008'.      I/S 8
G02IS   ENTIT 1-40-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-41-X'0000000E'.  CPC 14
G02IID  ENTIT 1-41-X'00000009'.      I/S 9
G02IS   ENTIT 1-41-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'0000000E'.  CPC 14
G02IID  ENTIT 1-42-X'0000000A'.      I/S 10
G02IS   ENTIT 1-42-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'0000000E'.  CPC 14
G02IID  ENTIT 1-43-X'0000000B'.      I/S 11
G02IS   ENTIT 1-43-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'0000000E'.  CPC 14
G02IID  ENTIT 1-44-X'0000000C'.      I/S 12
G02IS   ENTIT 1-44-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-45-X'0000000E'.  CPC 14
G02IID  ENTIT 1-45-X'0000000D'.      I/S 13

```

```

G02IS  ENTIT 1-45-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'0000000E'.          CPC 14
G02IID  ENTIT 1-46-X'0000000E'.          I/S 14
G02IS  ENTIT 1-46-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'0000000E'.          CPC 14
G02IID  ENTIT 1-47-X'0000000F'.          I/S 15
G02IS  ENTIT 1-47-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'0000000E'.          CPC 14
G02IID  ENTIT 1-48-X'00000010'.          I/S 16
G02IS  ENTIT 1-48-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-49-X'0000000F'.          CPC 15
G02IID  ENTIT 1-49-X'00000001'.          I/S 1
G02IS  ENTIT 1-49-X'00000001'.          GOA ORD NUM
*
G02CID  ENTIT 1-50-X'0000000F'.          CPC 15
G02IID  ENTIT 1-50-X'00000002'.          I/S 2
G02IS  ENTIT 1-50-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'0000000F'.          CPC 15
G02IID  ENTIT 1-51-X'00000003'.          I/S 3
G02IS  ENTIT 1-51-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'0000000F'.          CPC 15
G02IID  ENTIT 1-52-X'00000004'.          I/S 4
G02IS  ENTIT 1-52-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-53-X'0000000F'.          CPC 15
G02IID  ENTIT 1-53-X'00000005'.          I/S 5
G02IS  ENTIT 1-53-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'0000000F'.          CPC 15
G02IID  ENTIT 1-54-X'00000006'.          I/S 6
G02IS  ENTIT 1-54-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'0000000F'.          CPC 15
G02IID  ENTIT 1-55-X'00000007'.          I/S 7
G02IS  ENTIT 1-55-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'0000000F'.          CPC 15
G02IID  ENTIT 1-56-X'00000008'.          I/S 8
G02IS  ENTIT 1-56-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-57-X'0000000F'.          CPC 15
G02IID  ENTIT 1-57-X'00000009'.          I/S 9
G02IS  ENTIT 1-57-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'0000000F'.          CPC 15
G02IID  ENTIT 1-58-X'0000000A'.          I/S 10
G02IS  ENTIT 1-58-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'0000000F'.          CPC 15
G02IID  ENTIT 1-59-X'0000000B'.          I/S 11
G02IS  ENTIT 1-59-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'0000000F'.          CPC 15
G02IID  ENTIT 1-60-X'0000000C'.          I/S 12
G02IS  ENTIT 1-60-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'0000000F'.          CPC 15

```

```

G02IID  ENTIT 1-61-X'0000000D'.      I/S 13
G02IS   ENTIT 1-61-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'0000000F'.      CPC 15
G02IID  ENTIT 1-62-X'0000000E'.      I/S 14
G02IS   ENTIT 1-62-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'0000000F'.      CPC 15
G02IID  ENTIT 1-63-X'0000000F'.      I/S 15
G02IS   ENTIT 1-63-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'0000000F'.      CPC 15
G02IID  ENTIT 1-64-X'00000010'.      I/S 16
G02IS   ENTIT 1-64-X'00000010'.      GOA ORD NUM (#16)
*
*
*****
GEND
*****
* This is the 5th SUPERGOA ordinal
*
G01GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)93.      THIS IS #GLOBL 93 (x'5D')
G01BID  ENT  GO.              ID = 'GO'
G01CTL  ENT  X'00'.           N/A
G01FCH  ENT  X'005E'.         Chained to ordinal 94
G01NIS  ENTIT 1-1-X'0040'.    16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.      LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.      G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'.    CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'.    CPC 1 NOT IN THIS RECORD
G02DSP  ENTIT 1-03-X'8000'.    CPC 2 NOT IN THIS RECORD
G02DSP  ENTIT 1-04-X'8000'.    CPC 3 NOT IN THIS RECORD
G02DSP  ENTIT 1-05-X'8000'.    CPC 4 NOT IN THIS RECORD
G02DSP  ENTIT 1-06-X'8000'.    CPC 5 NOT IN THIS RECORD
G02DSP  ENTIT 1-07-X'8000'.    CPC 6 NOT IN THIS RECORD
G02DSP  ENTIT 1-08-X'8000'.    CPC 7 NOT IN THIS RECORD
G02DSP  ENTIT 1-09-X'8000'.    CPC 8 NOT IN THIS RECORD
G02DSP  ENTIT 1-10-X'8000'.    CPC 9 NOT IN THIS RECORD
G02DSP  ENTIT 1-11-X'8000'.    CPC 10 NOT IN THIS RECORD
G02DSP  ENTIT 1-12-X'8000'.    CPC 11 NOT IN THIS RECORD
G02DSP  ENTIT 1-13-X'8000'.    CPC 12 NOT IN THIS RECORD
G02DSP  ENTIT 1-14-X'8000'.    CPC 13 NOT IN THIS RECORD
G02DSP  ENTIT 1-15-X'8000'.    CPC 14 NOT IN THIS RECORD
G02DSP  ENTIT 1-16-X'8000'.    CPC 15 NOT IN THIS RECORD
G02DSP  ENTIT 1-17-X'0000'.    CPC 16 - 16 I STREAMS
G02DSP  ENTIT 1-18-X'0010'.    CPC 17 - 16 I STREAMS
G02DSP  ENTIT 1-19-X'0020'.    CPC 18 - 16 I STREAMS
G02DSP  ENTIT 1-20-X'0030'.    CPC 19 - 16 I STREAMS
G02DSP  ENTIT 1-21-X'8000'.    CPC 20 NOT IN THIS RECORD
G02DSP  ENTIT 1-22-X'8000'.    CPC 21 NOT IN THIS RECORD
G02DSP  ENTIT 1-23-X'8000'.    CPC 22 NOT IN THIS RECORD
G02DSP  ENTIT 1-24-X'8000'.    CPC 23 NOT IN THIS RECORD
G02DSP  ENTIT 1-25-X'8000'.    CPC 24 NOT IN THIS RECORD
G02DSP  ENTIT 1-26-X'8000'.    CPC 25 NOT IN THIS RECORD
G02DSP  ENTIT 1-27-X'8000'.    CPC 26 NOT IN THIS RECORD
G02DSP  ENTIT 1-28-X'8000'.    CPC 27 NOT IN THIS RECORD
G02DSP  ENTIT 1-29-X'8000'.    CPC 28 NOT IN THIS RECORD
G02DSP  ENTIT 1-30-X'8000'.    CPC 29 NOT IN THIS RECORD
G02DSP  ENTIT 1-31-X'8000'.    CPC 30 NOT IN THIS RECORD
G02DSP  ENTIT 1-32-X'8000'.    CPC 31 NOT IN THIS RECORD
*
*
G02CID  ENTIT 1-01-X'00000010'.    CPC 16
G02IID  ENTIT 1-01-X'00000001'.    I/S 1
G02IS   ENTIT 1-01-X'00000001'.    GOA ORD NUM

```



```

*
G02CID ENTIT 1-02-X'00000010'. CPC 16
G02IID ENTIT 1-02-X'00000002'. I/S 2
G02IS ENTIT 1-02-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-03-X'00000010'. CPC 16
G02IID ENTIT 1-03-X'00000003'. I/S 3
G02IS ENTIT 1-03-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-04-X'00000010'. CPC 16
G02IID ENTIT 1-04-X'00000004'. I/S 4
G02IS ENTIT 1-04-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-05-X'00000010'. CPC 16
G02IID ENTIT 1-05-X'00000005'. I/S 5
G02IS ENTIT 1-05-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-06-X'00000010'. CPC 16
G02IID ENTIT 1-06-X'00000006'. I/S 6
G02IS ENTIT 1-06-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-07-X'00000010'. CPC 16
G02IID ENTIT 1-07-X'00000007'. I/S 7
G02IS ENTIT 1-07-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-08-X'00000010'. CPC 16
G02IID ENTIT 1-08-X'00000008'. I/S 8
G02IS ENTIT 1-08-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-09-X'00000010'. CPC 16
G02IID ENTIT 1-09-X'00000009'. I/S 9
G02IS ENTIT 1-09-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-10-X'00000010'. CPC 16
G02IID ENTIT 1-10-X'0000000A'. I/S 10
G02IS ENTIT 1-10-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-11-X'00000010'. CPC 16
G02IID ENTIT 1-11-X'0000000B'. I/S 11
G02IS ENTIT 1-11-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-12-X'00000010'. CPC 16
G02IID ENTIT 1-12-X'0000000C'. I/S 12
G02IS ENTIT 1-12-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-13-X'00000010'. CPC 16
G02IID ENTIT 1-13-X'0000000D'. I/S 13
G02IS ENTIT 1-13-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-14-X'00000010'. CPC 16
G02IID ENTIT 1-14-X'0000000E'. I/S 14
G02IS ENTIT 1-14-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-15-X'00000010'. CPC 16
G02IID ENTIT 1-15-X'0000000F'. I/S 15
G02IS ENTIT 1-15-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-16-X'00000010'. CPC 16
G02IID ENTIT 1-16-X'00000010'. I/S 16
G02IS ENTIT 1-16-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-17-X'00000011'. CPC 17
G02IID ENTIT 1-17-X'00000001'. I/S 1

```

```

G02IS  ENTIT 1-17-X'00000001'.          GOA ORD NUM
*
G02CID  ENTIT 1-18-X'00000011'.  CPC 17
G02IID  ENTIT 1-18-X'00000002'.          I/S 2
G02IS  ENTIT 1-18-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-19-X'00000011'.  CPC 17
G02IID  ENTIT 1-19-X'00000003'.          I/S 3
G02IS  ENTIT 1-19-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-20-X'00000011'.  CPC 17
G02IID  ENTIT 1-20-X'00000004'.          I/S 4
G02IS  ENTIT 1-20-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-21-X'00000011'.  CPC 17
G02IID  ENTIT 1-21-X'00000005'.          I/S 5
G02IS  ENTIT 1-21-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-22-X'00000011'.  CPC 17
G02IID  ENTIT 1-22-X'00000006'.          I/S 6
G02IS  ENTIT 1-22-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-23-X'00000011'.  CPC 17
G02IID  ENTIT 1-23-X'00000007'.          I/S 7
G02IS  ENTIT 1-23-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-24-X'00000011'.  CPC 17
G02IID  ENTIT 1-24-X'00000008'.          I/S 8
G02IS  ENTIT 1-24-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-25-X'00000011'.  CPC 17
G02IID  ENTIT 1-25-X'00000009'.          I/S 9
G02IS  ENTIT 1-25-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-26-X'00000011'.  CPC 17
G02IID  ENTIT 1-26-X'0000000A'.          I/S 10
G02IS  ENTIT 1-26-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-27-X'00000011'.  CPC 17
G02IID  ENTIT 1-27-X'0000000B'.          I/S 11
G02IS  ENTIT 1-27-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-28-X'00000011'.  CPC 17
G02IID  ENTIT 1-28-X'0000000C'.          I/S 12
G02IS  ENTIT 1-28-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-29-X'00000011'.  CPC 17
G02IID  ENTIT 1-29-X'0000000D'.          I/S 13
G02IS  ENTIT 1-29-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-30-X'00000011'.  CPC 17
G02IID  ENTIT 1-30-X'0000000E'.          I/S 14
G02IS  ENTIT 1-30-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-31-X'00000011'.  CPC 17
G02IID  ENTIT 1-31-X'0000000F'.          I/S 15
G02IS  ENTIT 1-31-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-32-X'00000011'.  CPC 17
G02IID  ENTIT 1-32-X'00000010'.          I/S 16
G02IS  ENTIT 1-32-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-33-X'00000012'.  CPC 18

```

G02IID	ENTIT 1-33-X'00000001'.	I/S 1	
G02IS	ENTIT 1-33-X'00000001'.		GOA ORD NUM
*			
G02CID	ENTIT 1-34-X'00000012'.	CPC 18	
G02IID	ENTIT 1-34-X'00000002'.	I/S 2	
G02IS	ENTIT 1-34-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-35-X'00000012'.	CPC 18	
G02IID	ENTIT 1-35-X'00000003'.	I/S 3	
G02IS	ENTIT 1-35-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-36-X'00000012'.	CPC 18	
G02IID	ENTIT 1-36-X'00000004'.	I/S 4	
G02IS	ENTIT 1-36-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-37-X'00000012'.	CPC 18	
G02IID	ENTIT 1-37-X'00000005'.	I/S 5	
G02IS	ENTIT 1-37-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-38-X'00000012'.	CPC 18	
G02IID	ENTIT 1-38-X'00000006'.	I/S 6	
G02IS	ENTIT 1-38-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-39-X'00000012'.	CPC 18	
G02IID	ENTIT 1-39-X'00000007'.	I/S 7	
G02IS	ENTIT 1-39-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-40-X'00000012'.	CPC 18	
G02IID	ENTIT 1-40-X'00000008'.	I/S 8	
G02IS	ENTIT 1-40-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-41-X'00000012'.	CPC 18	
G02IID	ENTIT 1-41-X'00000009'.	I/S 9	
G02IS	ENTIT 1-41-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-42-X'00000012'.	CPC 18	
G02IID	ENTIT 1-42-X'0000000A'.	I/S 10	
G02IS	ENTIT 1-42-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-43-X'00000012'.	CPC 18	
G02IID	ENTIT 1-43-X'0000000B'.	I/S 11	
G02IS	ENTIT 1-43-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-44-X'00000012'.	CPC 18	
G02IID	ENTIT 1-44-X'0000000C'.	I/S 12	
G02IS	ENTIT 1-44-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-45-X'00000012'.	CPC 18	
G02IID	ENTIT 1-45-X'0000000D'.	I/S 13	
G02IS	ENTIT 1-45-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-46-X'00000012'.	CPC 18	
G02IID	ENTIT 1-46-X'0000000E'.	I/S 14	
G02IS	ENTIT 1-46-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-47-X'00000012'.	CPC 18	
G02IID	ENTIT 1-47-X'0000000F'.	I/S 15	
G02IS	ENTIT 1-47-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-48-X'00000012'.	CPC 18	
G02IID	ENTIT 1-48-X'00000010'.	I/S 16	
G02IS	ENTIT 1-48-X'00000010'.		GOA ORD NUM (#16)
*			

*			

```

G02CID  ENTIT 1-49-X'00000013'.  CPC 19
G02IID  ENTIT 1-49-X'00000001'.      I/S 1
G02IS   ENTIT 1-49-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-50-X'00000013'.  CPC 19
G02IID  ENTIT 1-50-X'00000002'.      I/S 2
G02IS   ENTIT 1-50-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'00000013'.  CPC 19
G02IID  ENTIT 1-51-X'00000003'.      I/S 3
G02IS   ENTIT 1-51-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'00000013'.  CPC 19
G02IID  ENTIT 1-52-X'00000004'.      I/S 4
G02IS   ENTIT 1-52-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-53-X'00000013'.  CPC 19
G02IID  ENTIT 1-53-X'00000005'.      I/S 5
G02IS   ENTIT 1-53-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'00000013'.  CPC 19
G02IID  ENTIT 1-54-X'00000006'.      I/S 6
G02IS   ENTIT 1-54-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'00000013'.  CPC 19
G02IID  ENTIT 1-55-X'00000007'.      I/S 7
G02IS   ENTIT 1-55-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'00000013'.  CPC 19
G02IID  ENTIT 1-56-X'00000008'.      I/S 8
G02IS   ENTIT 1-56-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-57-X'00000013'.  CPC 19
G02IID  ENTIT 1-57-X'00000009'.      I/S 9
G02IS   ENTIT 1-57-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'00000013'.  CPC 19
G02IID  ENTIT 1-58-X'0000000A'.      I/S 10
G02IS   ENTIT 1-58-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'00000013'.  CPC 19
G02IID  ENTIT 1-59-X'0000000B'.      I/S 11
G02IS   ENTIT 1-59-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'00000013'.  CPC 19
G02IID  ENTIT 1-60-X'0000000C'.      I/S 12
G02IS   ENTIT 1-60-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'00000013'.  CPC 19
G02IID  ENTIT 1-61-X'0000000D'.      I/S 13
G02IS   ENTIT 1-61-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'00000013'.  CPC 19
G02IID  ENTIT 1-62-X'0000000E'.      I/S 14
G02IS   ENTIT 1-62-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'00000013'.  CPC 19
G02IID  ENTIT 1-63-X'0000000F'.      I/S 15
G02IS   ENTIT 1-63-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'00000013'.  CPC 19
G02IID  ENTIT 1-64-X'00000010'.      I/S 16
G02IS   ENTIT 1-64-X'00000010'.      GOA ORD NUM (#16)
*
*

```

```

*****
GEND
*****
* This is the 6th SUPERGOA ordinal
*
G01GO    GSTAR 1.
BSTA06   ENT   (#GLOBL)94.      THIS IS #GLOBL 93 (x'5E')
G01BID   ENT   GO.              ID = 'GO'
G01CTL   ENT   X'00'.           N/A
G01FCH   ENT   X'005F'.         Chained to ordinal 95
G01NIS   ENTIT 1-1-X'0040'.     16 I/STREAMS * 4 CPCs
G01NUM   ENTIT 1-1-X'00'.       LOAD MODE N/A
G01CHN   ENTIT 1-1-X'40'.       G02DSP Entries Used
*
G02DSP   ENTIT 1-01-X'8000'.    CPC 0 NOT IN THIS RECORD
G02DSP   ENTIT 1-02-X'8000'.    CPC 1 NOT IN THIS RECORD
G02DSP   ENTIT 1-03-X'8000'.    CPC 2 NOT IN THIS RECORD
G02DSP   ENTIT 1-04-X'8000'.    CPC 3 NOT IN THIS RECORD
G02DSP   ENTIT 1-05-X'8000'.    CPC 4 NOT IN THIS RECORD
G02DSP   ENTIT 1-06-X'8000'.    CPC 5 NOT IN THIS RECORD
G02DSP   ENTIT 1-07-X'8000'.    CPC 6 NOT IN THIS RECORD
G02DSP   ENTIT 1-08-X'8000'.    CPC 7 NOT IN THIS RECORD
G02DSP   ENTIT 1-09-X'8000'.    CPC 8 NOT IN THIS RECORD
G02DSP   ENTIT 1-10-X'8000'.    CPC 9 NOT IN THIS RECORD
G02DSP   ENTIT 1-11-X'8000'.    CPC 10 NOT IN THIS RECORD
G02DSP   ENTIT 1-12-X'8000'.    CPC 11 NOT IN THIS RECORD
G02DSP   ENTIT 1-13-X'8000'.    CPC 12 NOT IN THIS RECORD
G02DSP   ENTIT 1-14-X'8000'.    CPC 13 NOT IN THIS RECORD
G02DSP   ENTIT 1-15-X'8000'.    CPC 14 NOT IN THIS RECORD
G02DSP   ENTIT 1-16-X'8000'.    CPC 15 NOT IN THIS RECORD
G02DSP   ENTIT 1-17-X'8000'.    CPC 16 NOT IN THIS RECORD
G02DSP   ENTIT 1-18-X'8000'.    CPC 17 NOT IN THIS RECORD
G02DSP   ENTIT 1-19-X'8000'.    CPC 18 NOT IN THIS RECORD
G02DSP   ENTIT 1-20-X'8000'.    CPC 19 NOT IN THIS RECORD
G02DSP   ENTIT 1-21-X'0000'.    CPC 20 - 16 I STREAMS
G02DSP   ENTIT 1-22-X'0010'.    CPC 21 - 16 I STREAMS
G02DSP   ENTIT 1-23-X'0020'.    CPC 22 - 16 I STREAMS
G02DSP   ENTIT 1-24-X'0030'.    CPC 23 - 16 I STREAMS
G02DSP   ENTIT 1-25-X'8000'.    CPC 24 NOT IN THIS RECORD
G02DSP   ENTIT 1-26-X'8000'.    CPC 25 NOT IN THIS RECORD
G02DSP   ENTIT 1-27-X'8000'.    CPC 26 NOT IN THIS RECORD
G02DSP   ENTIT 1-28-X'8000'.    CPC 27 NOT IN THIS RECORD
G02DSP   ENTIT 1-29-X'8000'.    CPC 28 NOT IN THIS RECORD
G02DSP   ENTIT 1-30-X'8000'.    CPC 29 NOT IN THIS RECORD
G02DSP   ENTIT 1-31-X'8000'.    CPC 30 NOT IN THIS RECORD
G02DSP   ENTIT 1-32-X'8000'.    CPC 31 NOT IN THIS RECORD
*
*
G02CID   ENTIT 1-01-X'00000014'. CPC 20
G02IID   ENTIT 1-01-X'00000001'.    I/S 1
G02IS    ENTIT 1-01-X'00000001'.    GOA ORD NUM
*
G02CID   ENTIT 1-02-X'00000014'.    CPC 20
G02IID   ENTIT 1-02-X'00000002'.    I/S 2
G02IS    ENTIT 1-02-X'00000010'.    GOA ORD NUM (#16)
*
G02CID   ENTIT 1-03-X'00000014'.    CPC 20
G02IID   ENTIT 1-03-X'00000003'.    I/S 3
G02IS    ENTIT 1-03-X'00000010'.    GOA ORD NUM (#16)
*
G02CID   ENTIT 1-04-X'00000014'.    CPC 20
G02IID   ENTIT 1-04-X'00000004'.    I/S 4
G02IS    ENTIT 1-04-X'00000010'.    GOA ORD NUM (#16)
*
G02CID   ENTIT 1-05-X'00000014'.    CPC 20
G02IID   ENTIT 1-05-X'00000005'.    I/S 5
G02IS    ENTIT 1-05-X'00000010'.    GOA ORD NUM (#16)

```

```

*
G02CID ENTIT 1-06-X'00000014'. CPC 20
G02IID ENTIT 1-06-X'00000006'. I/S 6
G02IS ENTIT 1-06-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-07-X'00000014'. CPC 20
G02IID ENTIT 1-07-X'00000007'. I/S 7
G02IS ENTIT 1-07-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-08-X'00000014'. CPC 20
G02IID ENTIT 1-08-X'00000008'. I/S 8
G02IS ENTIT 1-08-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-09-X'00000014'. CPC 20
G02IID ENTIT 1-09-X'00000009'. I/S 9
G02IS ENTIT 1-09-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-10-X'00000014'. CPC 20
G02IID ENTIT 1-10-X'0000000A'. I/S 10
G02IS ENTIT 1-10-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-11-X'00000014'. CPC 20
G02IID ENTIT 1-11-X'0000000B'. I/S 11
G02IS ENTIT 1-11-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-12-X'00000014'. CPC 20
G02IID ENTIT 1-12-X'0000000C'. I/S 12
G02IS ENTIT 1-12-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-13-X'00000014'. CPC 20
G02IID ENTIT 1-13-X'0000000D'. I/S 13
G02IS ENTIT 1-13-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-14-X'00000014'. CPC 20
G02IID ENTIT 1-14-X'0000000E'. I/S 14
G02IS ENTIT 1-14-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-15-X'00000014'. CPC 20
G02IID ENTIT 1-15-X'0000000F'. I/S 15
G02IS ENTIT 1-15-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-16-X'00000014'. CPC 20
G02IID ENTIT 1-16-X'00000010'. I/S 16
G02IS ENTIT 1-16-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-17-X'00000015'. CPC 21
G02IID ENTIT 1-17-X'00000001'. I/S 1
G02IS ENTIT 1-17-X'00000001'. GOA ORD NUM
*
G02CID ENTIT 1-18-X'00000015'. CPC 21
G02IID ENTIT 1-18-X'00000002'. I/S 2
G02IS ENTIT 1-18-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-19-X'00000015'. CPC 21
G02IID ENTIT 1-19-X'00000003'. I/S 3
G02IS ENTIT 1-19-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-20-X'00000015'. CPC 21
G02IID ENTIT 1-20-X'00000004'. I/S 4
G02IS ENTIT 1-20-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-21-X'00000015'. CPC 21
G02IID ENTIT 1-21-X'00000005'. I/S 5

```

```

G02IS  ENTIT 1-21-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-22-X'00000015'.          CPC 21
G02IID  ENTIT 1-22-X'00000006'.          I/S 6
G02IS   ENTIT 1-22-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-23-X'00000015'.          CPC 21
G02IID  ENTIT 1-23-X'00000007'.          I/S 7
G02IS   ENTIT 1-23-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-24-X'00000015'.          CPC 21
G02IID  ENTIT 1-24-X'00000008'.          I/S 8
G02IS   ENTIT 1-24-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-25-X'00000015'.          CPC 21
G02IID  ENTIT 1-25-X'00000009'.          I/S 9
G02IS   ENTIT 1-25-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-26-X'00000015'.          CPC 21
G02IID  ENTIT 1-26-X'0000000A'.          I/S 10
G02IS   ENTIT 1-26-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-27-X'00000015'.          CPC 21
G02IID  ENTIT 1-27-X'0000000B'.          I/S 11
G02IS   ENTIT 1-27-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-28-X'00000015'.          CPC 21
G02IID  ENTIT 1-28-X'0000000C'.          I/S 12
G02IS   ENTIT 1-28-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-29-X'00000015'.          CPC 21
G02IID  ENTIT 1-29-X'0000000D'.          I/S 13
G02IS   ENTIT 1-29-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-30-X'00000015'.          CPC 21
G02IID  ENTIT 1-30-X'0000000E'.          I/S 14
G02IS   ENTIT 1-30-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-31-X'00000015'.          CPC 21
G02IID  ENTIT 1-31-X'0000000F'.          I/S 15
G02IS   ENTIT 1-31-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-32-X'00000015'.          CPC 21
G02IID  ENTIT 1-32-X'00000010'.          I/S 16
G02IS   ENTIT 1-32-X'00000010'.          GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-33-X'00000016'.          CPC 22
G02IID  ENTIT 1-33-X'00000001'.          I/S 1
G02IS   ENTIT 1-33-X'00000001'.          GOA ORD NUM
*
G02CID  ENTIT 1-34-X'00000016'.          CPC 22
G02IID  ENTIT 1-34-X'00000002'.          I/S 2
G02IS   ENTIT 1-34-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-35-X'00000016'.          CPC 22
G02IID  ENTIT 1-35-X'00000003'.          I/S 3
G02IS   ENTIT 1-35-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-36-X'00000016'.          CPC 22
G02IID  ENTIT 1-36-X'00000004'.          I/S 4
G02IS   ENTIT 1-36-X'00000010'.          GOA ORD NUM (#16)
*
G02CID  ENTIT 1-37-X'00000016'.          CPC 22

```

```

G02IID  ENTIT 1-37-X'00000005'.      I/S 5
G02IS   ENTIT 1-37-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'00000016'.      CPC 22
G02IID  ENTIT 1-38-X'00000006'.      I/S 6
G02IS   ENTIT 1-38-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'00000016'.      CPC 22
G02IID  ENTIT 1-39-X'00000007'.      I/S 7
G02IS   ENTIT 1-39-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'00000016'.      CPC 22
G02IID  ENTIT 1-40-X'00000008'.      I/S 8
G02IS   ENTIT 1-40-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-41-X'00000016'.      CPC 22
G02IID  ENTIT 1-41-X'00000009'.      I/S 9
G02IS   ENTIT 1-41-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'00000016'.      CPC 22
G02IID  ENTIT 1-42-X'0000000A'.      I/S 10
G02IS   ENTIT 1-42-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'00000016'.      CPC 22
G02IID  ENTIT 1-43-X'0000000B'.      I/S 11
G02IS   ENTIT 1-43-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'00000016'.      CPC 22
G02IID  ENTIT 1-44-X'0000000C'.      I/S 12
G02IS   ENTIT 1-44-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-45-X'00000016'.      CPC 22
G02IID  ENTIT 1-45-X'0000000D'.      I/S 13
G02IS   ENTIT 1-45-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'00000016'.      CPC 22
G02IID  ENTIT 1-46-X'0000000E'.      I/S 14
G02IS   ENTIT 1-46-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'00000016'.      CPC 22
G02IID  ENTIT 1-47-X'0000000F'.      I/S 15
G02IS   ENTIT 1-47-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'00000016'.      CPC 22
G02IID  ENTIT 1-48-X'00000010'.      I/S 16
G02IS   ENTIT 1-48-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-49-X'00000017'.      CPC 23
G02IID  ENTIT 1-49-X'00000001'.      I/S 1
G02IS   ENTIT 1-49-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-50-X'00000017'.      CPC 23
G02IID  ENTIT 1-50-X'00000002'.      I/S 2
G02IS   ENTIT 1-50-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'00000017'.      CPC 23
G02IID  ENTIT 1-51-X'00000003'.      I/S 3
G02IS   ENTIT 1-51-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'00000017'.      CPC 23
G02IID  ENTIT 1-52-X'00000004'.      I/S 4
G02IS   ENTIT 1-52-X'00000010'.      GOA ORD NUM (#16)
*

```



```

G02CID  ENTIT 1-53-X'00000017'.  CPC 23
G02IID  ENTIT 1-53-X'00000005'.      I/S 5
G02IS   ENTIT 1-53-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'00000017'.  CPC 23
G02IID  ENTIT 1-54-X'00000006'.      I/S 6
G02IS   ENTIT 1-54-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'00000017'.  CPC 23
G02IID  ENTIT 1-55-X'00000007'.      I/S 7
G02IS   ENTIT 1-55-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'00000017'.  CPC 23
G02IID  ENTIT 1-56-X'00000008'.      I/S 8
G02IS   ENTIT 1-56-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-57-X'00000017'.  CPC 23
G02IID  ENTIT 1-57-X'00000009'.      I/S 9
G02IS   ENTIT 1-57-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'00000017'.  CPC 23
G02IID  ENTIT 1-58-X'0000000A'.      I/S 10
G02IS   ENTIT 1-58-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'00000017'.  CPC 23
G02IID  ENTIT 1-59-X'0000000B'.      I/S 11
G02IS   ENTIT 1-59-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'00000017'.  CPC 23
G02IID  ENTIT 1-60-X'0000000C'.      I/S 12
G02IS   ENTIT 1-60-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'00000017'.  CPC 23
G02IID  ENTIT 1-61-X'0000000D'.      I/S 13
G02IS   ENTIT 1-61-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'00000017'.  CPC 23
G02IID  ENTIT 1-62-X'0000000E'.      I/S 14
G02IS   ENTIT 1-62-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'00000017'.  CPC 23
G02IID  ENTIT 1-63-X'0000000F'.      I/S 15
G02IS   ENTIT 1-63-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'00000017'.  CPC 23
G02IID  ENTIT 1-64-X'00000010'.      I/S 16
G02IS   ENTIT 1-64-X'00000010'.      GOA ORD NUM (#16)
*
*
*****
      GEND
*****
* This is the 7th SUPERGOA ordinal
*
G01GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)95.      THIS IS #GLOBL 95 (x'5F')
G01BID  ENT  GO.              ID = 'GO'
G01CTL  ENT  X'00'.           N/A
G01FCH  ENT  X'0060'.         Chained to ordinal 96
G01NIS  ENTIT 1-1-X'0040'.     16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.       LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.       G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'.     CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'.     CPC 1 NOT IN THIS RECORD

```

G02DSP	ENTIT 1-03-X'8000'.	CPC 2 NOT IN THIS RECORD
G02DSP	ENTIT 1-04-X'8000'.	CPC 3 NOT IN THIS RECORD
G02DSP	ENTIT 1-05-X'8000'.	CPC 4 NOT IN THIS RECORD
G02DSP	ENTIT 1-06-X'8000'.	CPC 5 NOT IN THIS RECORD
G02DSP	ENTIT 1-07-X'8000'.	CPC 6 NOT IN THIS RECORD
G02DSP	ENTIT 1-08-X'8000'.	CPC 7 NOT IN THIS RECORD
G02DSP	ENTIT 1-09-X'8000'.	CPC 8 NOT IN THIS RECORD
G02DSP	ENTIT 1-10-X'8000'.	CPC 9 NOT IN THIS RECORD
G02DSP	ENTIT 1-11-X'8000'.	CPC 10 NOT IN THIS RECORD
G02DSP	ENTIT 1-12-X'8000'.	CPC 11 NOT IN THIS RECORD
G02DSP	ENTIT 1-13-X'8000'.	CPC 12 NOT IN THIS RECORD
G02DSP	ENTIT 1-14-X'8000'.	CPC 13 NOT IN THIS RECORD
G02DSP	ENTIT 1-15-X'8000'.	CPC 14 NOT IN THIS RECORD
G02DSP	ENTIT 1-16-X'8000'.	CPC 15 NOT IN THIS RECORD
G02DSP	ENTIT 1-17-X'8000'.	CPC 16 NOT IN THIS RECORD
G02DSP	ENTIT 1-18-X'8000'.	CPC 17 NOT IN THIS RECORD
G02DSP	ENTIT 1-19-X'8000'.	CPC 18 NOT IN THIS RECORD
G02DSP	ENTIT 1-20-X'8000'.	CPC 19 NOT IN THIS RECORD
G02DSP	ENTIT 1-21-X'8000'.	CPC 20 NOT IN THIS RECORD
G02DSP	ENTIT 1-22-X'8000'.	CPC 21 NOT IN THIS RECORD
G02DSP	ENTIT 1-23-X'8000'.	CPC 22 NOT IN THIS RECORD
G02DSP	ENTIT 1-24-X'8000'.	CPC 23 NOT IN THIS RECORD
G02DSP	ENTIT 1-25-X'0000'.	CPC 24
G02DSP	ENTIT 1-26-X'0010'.	CPC 25
G02DSP	ENTIT 1-27-X'0020'.	CPC 26
G02DSP	ENTIT 1-28-X'0030'.	CPC 27
G02DSP	ENTIT 1-29-X'8000'.	CPC 28 NOT IN THIS RECORD
G02DSP	ENTIT 1-30-X'8000'.	CPC 29 NOT IN THIS RECORD
G02DSP	ENTIT 1-31-X'8000'.	CPC 30 NOT IN THIS RECORD
G02DSP	ENTIT 1-32-X'8000'.	CPC 31 NOT IN THIS RECORD
*		
*		
G02CID	ENTIT 1-01-X'00000018'.	CPC 24
G02IID	ENTIT 1-01-X'00000001'.	I/S 1
G02IS	ENTIT 1-01-X'00000001'.	GOA ORD NUM
*		
G02CID	ENTIT 1-02-X'00000018'.	CPC 24
G02IID	ENTIT 1-02-X'00000002'.	I/S 2
G02IS	ENTIT 1-02-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-03-X'00000018'.	CPC 24
G02IID	ENTIT 1-03-X'00000003'.	I/S 3
G02IS	ENTIT 1-03-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-04-X'00000018'.	CPC 24
G02IID	ENTIT 1-04-X'00000004'.	I/S 4
G02IS	ENTIT 1-04-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-05-X'00000018'.	CPC 24
G02IID	ENTIT 1-05-X'00000005'.	I/S 5
G02IS	ENTIT 1-05-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-06-X'00000018'.	CPC 24
G02IID	ENTIT 1-06-X'00000006'.	I/S 6
G02IS	ENTIT 1-06-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-07-X'00000018'.	CPC 24
G02IID	ENTIT 1-07-X'00000007'.	I/S 7
G02IS	ENTIT 1-07-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-08-X'00000018'.	CPC 24
G02IID	ENTIT 1-08-X'00000008'.	I/S 8
G02IS	ENTIT 1-08-X'00000010'.	GOA ORD NUM (#16)
*		

*		
G02CID	ENTIT 1-09-X'00000018'.	CPC 24

G02IID	ENTIT 1-09-X'00000009'.	I/S 9	
G02IS	ENTIT 1-09-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-10-X'00000018'.	CPC 24	
G02IID	ENTIT 1-10-X'0000000A'.	I/S 10	
G02IS	ENTIT 1-10-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-11-X'00000018'.	CPC 24	
G02IID	ENTIT 1-11-X'0000000B'.	I/S 11	
G02IS	ENTIT 1-11-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-12-X'00000018'.	CPC 24	
G02IID	ENTIT 1-12-X'0000000C'.	I/S 12	
G02IS	ENTIT 1-12-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-13-X'00000018'.	CPC 24	
G02IID	ENTIT 1-13-X'0000000D'.	I/S 13	
G02IS	ENTIT 1-13-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-14-X'00000018'.	CPC 24	
G02IID	ENTIT 1-14-X'0000000E'.	I/S 14	
G02IS	ENTIT 1-14-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-15-X'00000018'.	CPC 24	
G02IID	ENTIT 1-15-X'0000000F'.	I/S 15	
G02IS	ENTIT 1-15-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-16-X'00000018'.	CPC 24	
G02IID	ENTIT 1-16-X'00000010'.	I/S 16	
G02IS	ENTIT 1-16-X'00000010'.		GOA ORD NUM (#16)
*			

*			
G02CID	ENTIT 1-17-X'00000019'.	CPC 25	
G02IID	ENTIT 1-17-X'00000001'.	I/S 1	
G02IS	ENTIT 1-17-X'00000001'.		GOA ORD NUM
*			
G02CID	ENTIT 1-18-X'00000019'.	CPC 25	
G02IID	ENTIT 1-18-X'00000002'.	I/S 2	
G02IS	ENTIT 1-18-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-19-X'00000019'.	CPC 25	
G02IID	ENTIT 1-19-X'00000003'.	I/S 3	
G02IS	ENTIT 1-19-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-20-X'00000019'.	CPC 25	
G02IID	ENTIT 1-20-X'00000004'.	I/S 4	
G02IS	ENTIT 1-20-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-21-X'00000019'.	CPC 25	
G02IID	ENTIT 1-21-X'00000005'.	I/S 5	
G02IS	ENTIT 1-21-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-22-X'00000019'.	CPC 25	
G02IID	ENTIT 1-22-X'00000006'.	I/S 6	
G02IS	ENTIT 1-22-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-23-X'00000019'.	CPC 25	
G02IID	ENTIT 1-23-X'00000007'.	I/S 7	
G02IS	ENTIT 1-23-X'00000010'.		GOA ORD NUM (#16)
*			
G02CID	ENTIT 1-24-X'00000019'.	CPC 25	
G02IID	ENTIT 1-24-X'00000008'.	I/S 8	
G02IS	ENTIT 1-24-X'00000010'.		GOA ORD NUM (#16)
*			

*			

```

G02CID  ENTIT 1-25-X'00000019'.  CPC 25
G02IID  ENTIT 1-25-X'00000009'.      I/S 9
G02IS   ENTIT 1-25-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-26-X'00000019'.  CPC 25
G02IID  ENTIT 1-26-X'0000000A'.      I/S 10
G02IS   ENTIT 1-26-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-27-X'00000019'.  CPC 25
G02IID  ENTIT 1-27-X'0000000B'.      I/S 11
G02IS   ENTIT 1-27-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-28-X'00000019'.  CPC 25
G02IID  ENTIT 1-28-X'0000000C'.      I/S 12
G02IS   ENTIT 1-28-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-29-X'00000019'.  CPC 25
G02IID  ENTIT 1-29-X'0000000D'.      I/S 13
G02IS   ENTIT 1-29-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-30-X'00000019'.  CPC 25
G02IID  ENTIT 1-30-X'0000000E'.      I/S 14
G02IS   ENTIT 1-30-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-31-X'00000019'.  CPC 25
G02IID  ENTIT 1-31-X'0000000F'.      I/S 15
G02IS   ENTIT 1-31-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-32-X'00000019'.  CPC 25
G02IID  ENTIT 1-32-X'00000010'.      I/S 16
G02IS   ENTIT 1-32-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-33-X'0000001A'.  CPC 26
G02IID  ENTIT 1-33-X'00000001'.      I/S 1
G02IS   ENTIT 1-33-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-34-X'0000001A'.  CPC 26
G02IID  ENTIT 1-34-X'00000002'.      I/S 2
G02IS   ENTIT 1-34-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-35-X'0000001A'.  CPC 26
G02IID  ENTIT 1-35-X'00000003'.      I/S 3
G02IS   ENTIT 1-35-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-36-X'0000001A'.  CPC 26
G02IID  ENTIT 1-36-X'00000004'.      I/S 4
G02IS   ENTIT 1-36-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-37-X'0000001A'.  CPC 26
G02IID  ENTIT 1-37-X'00000005'.      I/S 5
G02IS   ENTIT 1-37-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'0000001A'.  CPC 26
G02IID  ENTIT 1-38-X'00000006'.      I/S 6
G02IS   ENTIT 1-38-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'0000001A'.  CPC 26
G02IID  ENTIT 1-39-X'00000007'.      I/S 7
G02IS   ENTIT 1-39-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'0000001A'.  CPC 26
G02IID  ENTIT 1-40-X'00000008'.      I/S 8
G02IS   ENTIT 1-40-X'00000010'.      GOA ORD NUM (#16)
*
*****

```

```

*
G02CID ENTIT 1-41-X'0000001A'. CPC 26
G02IID ENTIT 1-41-X'00000009'. I/S 9
G02IS ENTIT 1-41-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-42-X'0000001A'. CPC 26
G02IID ENTIT 1-42-X'0000000A'. I/S 10
G02IS ENTIT 1-42-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-43-X'0000001A'. CPC 26
G02IID ENTIT 1-43-X'0000000B'. I/S 11
G02IS ENTIT 1-43-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-44-X'0000001A'. CPC 26
G02IID ENTIT 1-44-X'0000000C'. I/S 12
G02IS ENTIT 1-44-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-45-X'0000001A'. CPC 26
G02IID ENTIT 1-45-X'0000000D'. I/S 13
G02IS ENTIT 1-45-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-46-X'0000001A'. CPC 26
G02IID ENTIT 1-46-X'0000000E'. I/S 14
G02IS ENTIT 1-46-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-47-X'0000001A'. CPC 26
G02IID ENTIT 1-47-X'0000000F'. I/S 15
G02IS ENTIT 1-47-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-48-X'0000001A'. CPC 26
G02IID ENTIT 1-48-X'00000010'. I/S 16
G02IS ENTIT 1-48-X'00000010'. GOA ORD NUM (#16)
*
*****
*
G02CID ENTIT 1-49-X'0000001B'. CPC 27
G02IID ENTIT 1-49-X'00000001'. I/S 1
G02IS ENTIT 1-49-X'00000001'. GOA ORD NUM
*
G02CID ENTIT 1-50-X'0000001B'. CPC 27
G02IID ENTIT 1-50-X'00000002'. I/S 2
G02IS ENTIT 1-50-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-51-X'0000001B'. CPC 27
G02IID ENTIT 1-51-X'00000003'. I/S 3
G02IS ENTIT 1-51-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-52-X'0000001B'. CPC 27
G02IID ENTIT 1-52-X'00000004'. I/S 4
G02IS ENTIT 1-52-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-53-X'0000001B'. CPC 27
G02IID ENTIT 1-53-X'00000005'. I/S 5
G02IS ENTIT 1-53-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-54-X'0000001B'. CPC 27
G02IID ENTIT 1-54-X'00000006'. I/S 6
G02IS ENTIT 1-54-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-55-X'0000001B'. CPC 27
G02IID ENTIT 1-55-X'00000007'. I/S 7
G02IS ENTIT 1-55-X'00000010'. GOA ORD NUM (#16)
*
G02CID ENTIT 1-56-X'0000001B'. CPC 27
G02IID ENTIT 1-56-X'00000008'. I/S 8
G02IS ENTIT 1-56-X'00000010'. GOA ORD NUM (#16)
*

```

```

*****
*
G02CID  ENTIT 1-57-X'0000001B'.   CPC 27
G02IID  ENTIT 1-57-X'00000009'.   I/S 9
G02IS   ENTIT 1-57-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'0000001B'.   CPC 27
G02IID  ENTIT 1-58-X'0000000A'.   I/S 10
G02IS   ENTIT 1-58-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'0000001B'.   CPC 27
G02IID  ENTIT 1-59-X'0000000B'.   I/S 11
G02IS   ENTIT 1-59-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'0000001B'.   CPC 27
G02IID  ENTIT 1-60-X'0000000C'.   I/S 12
G02IS   ENTIT 1-60-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-61-X'0000001B'.   CPC 27
G02IID  ENTIT 1-61-X'0000000D'.   I/S 13
G02IS   ENTIT 1-61-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-62-X'0000001B'.   CPC 27
G02IID  ENTIT 1-62-X'0000000E'.   I/S 14
G02IS   ENTIT 1-62-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-63-X'0000001B'.   CPC 27
G02IID  ENTIT 1-63-X'0000000F'.   I/S 15
G02IS   ENTIT 1-63-X'00000010'.   GOA ORD NUM (#16)
*
G02CID  ENTIT 1-64-X'0000001B'.   CPC 27
G02IID  ENTIT 1-64-X'00000010'.   I/S 16
G02IS   ENTIT 1-64-X'00000010'.   GOA ORD NUM (#16)
*
*
*****
GEND
*****
* This is the 8th SUPERGOA ordinal
*
G01GO   GSTAR 1.
BSTA06  ENT  (#GLOBL)96.   THIS IS #GLOBL 96 (x'60')
G01BID  ENT  GO.          ID = 'GO'
G01CTL  ENT  X'00'.        N/A
G01FCH  ENT  X'0000'.      LAST IN CHAIN
G01NIS  ENTIT 1-1-X'0040'.  16 I/STREAMS * 4 CPCs
G01NUM  ENTIT 1-1-X'00'.    LOAD MODE N/A
G01CHN  ENTIT 1-1-X'40'.    G02DSP Entries Used
*
G02DSP  ENTIT 1-01-X'8000'.  CPC 0 NOT IN THIS RECORD
G02DSP  ENTIT 1-02-X'8000'.  CPC 1 NOT IN THIS RECORD
G02DSP  ENTIT 1-03-X'8000'.  CPC 2 NOT IN THIS RECORD
G02DSP  ENTIT 1-04-X'8000'.  CPC 3 NOT IN THIS RECORD
G02DSP  ENTIT 1-05-X'8000'.  CPC 4 NOT IN THIS RECORD
G02DSP  ENTIT 1-06-X'8000'.  CPC 5 NOT IN THIS RECORD
G02DSP  ENTIT 1-07-X'8000'.  CPC 6 NOT IN THIS RECORD
G02DSP  ENTIT 1-08-X'8000'.  CPC 7 NOT IN THIS RECORD
G02DSP  ENTIT 1-09-X'8000'.  CPC 8 NOT IN THIS RECORD
G02DSP  ENTIT 1-10-X'8000'.  CPC 9 NOT IN THIS RECORD
G02DSP  ENTIT 1-11-X'8000'.  CPC 10 NOT IN THIS RECORD
G02DSP  ENTIT 1-12-X'8000'.  CPC 11 NOT IN THIS RECORD
G02DSP  ENTIT 1-13-X'8000'.  CPC 12 NOT IN THIS RECORD
G02DSP  ENTIT 1-14-X'8000'.  CPC 13 NOT IN THIS RECORD
G02DSP  ENTIT 1-15-X'8000'.  CPC 14 NOT IN THIS RECORD
G02DSP  ENTIT 1-16-X'8000'.  CPC 15 NOT IN THIS RECORD
G02DSP  ENTIT 1-17-X'8000'.  CPC 16 NOT IN THIS RECORD
G02DSP  ENTIT 1-18-X'8000'.  CPC 17 NOT IN THIS RECORD

```

G02DSP	ENTIT 1-19-X'8000'.	CPC 18 NOT IN THIS RECORD
G02DSP	ENTIT 1-20-X'8000'.	CPC 19 NOT IN THIS RECORD
G02DSP	ENTIT 1-21-X'8000'.	CPC 20 NOT IN THIS RECORD
G02DSP	ENTIT 1-22-X'8000'.	CPC 21 NOT IN THIS RECORD
G02DSP	ENTIT 1-23-X'8000'.	CPC 22 NOT IN THIS RECORD
G02DSP	ENTIT 1-24-X'8000'.	CPC 23 NOT IN THIS RECORD
G02DSP	ENTIT 1-25-X'8000'.	CPC 24 NOT IN THIS RECORD
G02DSP	ENTIT 1-26-X'8000'.	CPC 25 NOT IN THIS RECORD
G02DSP	ENTIT 1-27-X'8000'.	CPC 26 NOT IN THIS RECORD
G02DSP	ENTIT 1-28-X'8000'.	CPC 27 NOT IN THIS RECORD
G02DSP	ENTIT 1-29-X'0000'.	CPC 28
G02DSP	ENTIT 1-30-X'0010'.	CPC 29
G02DSP	ENTIT 1-31-X'0020'.	CPC 30
G02DSP	ENTIT 1-32-X'0030'.	CPC 31
*		
*		
G02CID	ENTIT 1-01-X'0000001C'.	CPC 28
G02IID	ENTIT 1-01-X'00000001'.	I/S 1
G02IS	ENTIT 1-01-X'00000001'.	GOA ORD NUM
*		
G02CID	ENTIT 1-02-X'0000001C'.	CPC 28
G02IID	ENTIT 1-02-X'00000002'.	I/S 2
G02IS	ENTIT 1-02-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-03-X'0000001C'.	CPC 28
G02IID	ENTIT 1-03-X'00000003'.	I/S 3
G02IS	ENTIT 1-03-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-04-X'0000001C'.	CPC 28
G02IID	ENTIT 1-04-X'00000004'.	I/S 4
G02IS	ENTIT 1-04-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-05-X'0000001C'.	CPC 28
G02IID	ENTIT 1-05-X'00000005'.	I/S 5
G02IS	ENTIT 1-05-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-06-X'0000001C'.	CPC 28
G02IID	ENTIT 1-06-X'00000006'.	I/S 6
G02IS	ENTIT 1-06-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-07-X'0000001C'.	CPC 28
G02IID	ENTIT 1-07-X'00000007'.	I/S 7
G02IS	ENTIT 1-07-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-08-X'0000001C'.	CPC 28
G02IID	ENTIT 1-08-X'00000008'.	I/S 8
G02IS	ENTIT 1-08-X'00000010'.	GOA ORD NUM (#16)
*		

*		
G02CID	ENTIT 1-09-X'0000001C'.	CPC 28
G02IID	ENTIT 1-09-X'00000009'.	I/S 9
G02IS	ENTIT 1-09-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-10-X'0000001C'.	CPC 28
G02IID	ENTIT 1-10-X'0000000A'.	I/S 10
G02IS	ENTIT 1-10-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-11-X'0000001C'.	CPC 28
G02IID	ENTIT 1-11-X'0000000B'.	I/S 11
G02IS	ENTIT 1-11-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-12-X'0000001C'.	CPC 28
G02IID	ENTIT 1-12-X'0000000C'.	I/S 12
G02IS	ENTIT 1-12-X'00000010'.	GOA ORD NUM (#16)
*		
G02CID	ENTIT 1-13-X'0000001C'.	CPC 28

```

G02IID  ENTIT 1-13-X'0000000D'.      I/S 13
G02IS   ENTIT 1-13-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-14-X'0000001C'.      CPC 28
G02IID  ENTIT 1-14-X'0000000E'.      I/S 14
G02IS   ENTIT 1-14-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-15-X'0000001C'.      CPC 28
G02IID  ENTIT 1-15-X'0000000F'.      I/S 15
G02IS   ENTIT 1-15-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-16-X'0000001C'.      CPC 28
G02IID  ENTIT 1-16-X'00000010'.      I/S 16
G02IS   ENTIT 1-16-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-17-X'0000001D'.      CPC 29
G02IID  ENTIT 1-17-X'00000001'.      I/S 1
G02IS   ENTIT 1-17-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-18-X'0000001D'.      CPC 29
G02IID  ENTIT 1-18-X'00000002'.      I/S 2
G02IS   ENTIT 1-18-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-19-X'0000001D'.      CPC 29
G02IID  ENTIT 1-19-X'00000003'.      I/S 3
G02IS   ENTIT 1-19-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-20-X'0000001D'.      CPC 29
G02IID  ENTIT 1-20-X'00000004'.      I/S 4
G02IS   ENTIT 1-20-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-21-X'0000001D'.      CPC 29
G02IID  ENTIT 1-21-X'00000005'.      I/S 5
G02IS   ENTIT 1-21-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-22-X'0000001D'.      CPC 29
G02IID  ENTIT 1-22-X'00000006'.      I/S 6
G02IS   ENTIT 1-22-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-23-X'0000001D'.      CPC 29
G02IID  ENTIT 1-23-X'00000007'.      I/S 7
G02IS   ENTIT 1-23-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-24-X'0000001D'.      CPC 29
G02IID  ENTIT 1-24-X'00000008'.      I/S 8
G02IS   ENTIT 1-24-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-25-X'0000001D'.      CPC 29
G02IID  ENTIT 1-25-X'00000009'.      I/S 9
G02IS   ENTIT 1-25-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-26-X'0000001D'.      CPC 29
G02IID  ENTIT 1-26-X'0000000A'.      I/S 10
G02IS   ENTIT 1-26-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-27-X'0000001D'.      CPC 29
G02IID  ENTIT 1-27-X'0000000B'.      I/S 11
G02IS   ENTIT 1-27-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-28-X'0000001D'.      CPC 29
G02IID  ENTIT 1-28-X'0000000C'.      I/S 12
G02IS   ENTIT 1-28-X'00000010'.      GOA ORD NUM (#16)
*

```



```

G02CID  ENTIT 1-29-X'0000001D'.  CPC 29
G02IID  ENTIT 1-29-X'0000000D'.      I/S 13
G02IS   ENTIT 1-29-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-30-X'0000001D'.  CPC 29
G02IID  ENTIT 1-30-X'0000000E'.      I/S 14
G02IS   ENTIT 1-30-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-31-X'0000001D'.  CPC 29
G02IID  ENTIT 1-31-X'0000000F'.      I/S 15
G02IS   ENTIT 1-31-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-32-X'0000001D'.  CPC 29
G02IID  ENTIT 1-32-X'00000010'.      I/S 16
G02IS   ENTIT 1-32-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-33-X'0000001E'.  CPC 30
G02IID  ENTIT 1-33-X'00000001'.      I/S 1
G02IS   ENTIT 1-33-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-34-X'0000001E'.  CPC 30
G02IID  ENTIT 1-34-X'00000002'.      I/S 2
G02IS   ENTIT 1-34-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-35-X'0000001E'.  CPC 30
G02IID  ENTIT 1-35-X'00000003'.      I/S 3
G02IS   ENTIT 1-35-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-36-X'0000001E'.  CPC 30
G02IID  ENTIT 1-36-X'00000004'.      I/S 4
G02IS   ENTIT 1-36-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-37-X'0000001E'.  CPC 30
G02IID  ENTIT 1-37-X'00000005'.      I/S 5
G02IS   ENTIT 1-37-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-38-X'0000001E'.  CPC 30
G02IID  ENTIT 1-38-X'00000006'.      I/S 6
G02IS   ENTIT 1-38-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-39-X'0000001E'.  CPC 30
G02IID  ENTIT 1-39-X'00000007'.      I/S 7
G02IS   ENTIT 1-39-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-40-X'0000001E'.  CPC 30
G02IID  ENTIT 1-40-X'00000008'.      I/S 8
G02IS   ENTIT 1-40-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-41-X'0000001E'.  CPC 30
G02IID  ENTIT 1-41-X'00000009'.      I/S 9
G02IS   ENTIT 1-41-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-42-X'0000001E'.  CPC 30
G02IID  ENTIT 1-42-X'0000000A'.      I/S 10
G02IS   ENTIT 1-42-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-43-X'0000001E'.  CPC 30
G02IID  ENTIT 1-43-X'0000000B'.      I/S 11
G02IS   ENTIT 1-43-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-44-X'0000001E'.  CPC 30
G02IID  ENTIT 1-44-X'0000000C'.      I/S 12
G02IS   ENTIT 1-44-X'00000010'.      GOA ORD NUM (#16)

```

```

*
G02CID  ENTIT 1-45-X'0000001E'.  CPC 30
G02IID  ENTIT 1-45-X'0000000D'.      I/S 13
G02IS   ENTIT 1-45-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-46-X'0000001E'.  CPC 30
G02IID  ENTIT 1-46-X'0000000E'.      I/S 14
G02IS   ENTIT 1-46-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-47-X'0000001E'.  CPC 30
G02IID  ENTIT 1-47-X'0000000F'.      I/S 15
G02IS   ENTIT 1-47-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-48-X'0000001E'.  CPC 30
G02IID  ENTIT 1-48-X'00000010'.      I/S 16
G02IS   ENTIT 1-48-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-49-X'0000001F'.  CPC 31
G02IID  ENTIT 1-49-X'00000001'.      I/S 1
G02IS   ENTIT 1-49-X'00000001'.      GOA ORD NUM
*
G02CID  ENTIT 1-50-X'0000001F'.  CPC 31
G02IID  ENTIT 1-50-X'00000002'.      I/S 2
G02IS   ENTIT 1-50-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-51-X'0000001F'.  CPC 31
G02IID  ENTIT 1-51-X'00000003'.      I/S 3
G02IS   ENTIT 1-51-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-52-X'0000001F'.  CPC 31
G02IID  ENTIT 1-52-X'00000004'.      I/S 4
G02IS   ENTIT 1-52-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-53-X'0000001F'.  CPC 31
G02IID  ENTIT 1-53-X'00000005'.      I/S 5
G02IS   ENTIT 1-53-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-54-X'0000001F'.  CPC 31
G02IID  ENTIT 1-54-X'00000006'.      I/S 6
G02IS   ENTIT 1-54-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-55-X'0000001F'.  CPC 31
G02IID  ENTIT 1-55-X'00000007'.      I/S 7
G02IS   ENTIT 1-55-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-56-X'0000001F'.  CPC 31
G02IID  ENTIT 1-56-X'00000008'.      I/S 8
G02IS   ENTIT 1-56-X'00000010'.      GOA ORD NUM (#16)
*
*****
*
G02CID  ENTIT 1-57-X'0000001F'.  CPC 31
G02IID  ENTIT 1-57-X'00000009'.      I/S 9
G02IS   ENTIT 1-57-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-58-X'0000001F'.  CPC 31
G02IID  ENTIT 1-58-X'0000000A'.      I/S 10
G02IS   ENTIT 1-58-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-59-X'0000001F'.  CPC 31
G02IID  ENTIT 1-59-X'0000000B'.      I/S 11
G02IS   ENTIT 1-59-X'00000010'.      GOA ORD NUM (#16)
*
G02CID  ENTIT 1-60-X'0000001F'.  CPC 31
G02IID  ENTIT 1-60-X'0000000C'.      I/S 12

```

```

G02IS  ENTIT 1-60-X'00000010'.          GOA ORD NUM (#16)
*
G02CID ENTIT 1-61-X'0000001F'.          CPC 31
G02IID ENTIT 1-61-X'0000000D'.          I/S 13
G02IS  ENTIT 1-61-X'00000010'.          GOA ORD NUM (#16)
*
G02CID ENTIT 1-62-X'0000001F'.          CPC 31
G02IID ENTIT 1-62-X'0000000E'.          I/S 14
G02IS  ENTIT 1-62-X'00000010'.          GOA ORD NUM (#16)
*
G02CID ENTIT 1-63-X'0000001F'.          CPC 31
G02IID ENTIT 1-63-X'0000000F'.          I/S 15
G02IS  ENTIT 1-63-X'00000010'.          GOA ORD NUM (#16)
*
G02CID ENTIT 1-64-X'0000001F'.          CPC 31
G02IID ENTIT 1-64-X'00000010'.          I/S 16
G02IS  ENTIT 1-64-X'00000010'.          GOA ORD NUM (#16)
*
*
*****
GEND
*****

```

In the example that follows, if the CPC number is 0 and the I-stream number is 2, the ordinal number of the prime GOA for this CPC/I-stream pair would be 18 (that is, X'12'). Each GO2DSP field in section 3 should contain the total number of entries that precede the entry for this CPC in the table in section 4.

Following is an example of how the STC input might be coded for a super GOA:

```

*      SUPER GOA FOR 2 CPC'S, EACH WITH 2 I-STREAMS
G01GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)0.          THIS IS #GLOBL ORDINAL NUMBER 0
*      STANDARD HEADER
G01BID ENT  GO.                  ID = 'GO'
G01FCH ENT  X'00000000'.         N/A - NO FORWARD CHAIN
*      TABLE HEADER
G01NIS  ENTIT 1-1-X'0004'.        MAX # OF I-STREAMS IN LC COMPLEX
G01CHN  ENTIT 1-1-X'40'.         GO2DSP Entries Used
*
*      TABLE 1 - CPC INDEX INTO TABLE 2
G02DSP  ENTIT 1-1-X'0000'.        CPC 0 STARTS AT THE FIRST ENTRY
G02DSP  ENTIT 1-2-X'0002'.        CPC 1 STARTS AT THE SECOND ENTRY
*      TABLE 2 - INDEX TO I-STREAMS' PRIME GOA
G02CID  ENTIT 1-1-X'00000000'.    CPC 0
G02IID  ENTIT 1-1-X'00000001'.    I-STREAM 1
G02IS   ENTIT 1-1-X'00000001'.    PRIME GOA ORD NUM (#1)
G02CID  ENTIT 1-2-X'00000000'.    CPC 0
G02IID  ENTIT 1-2-X'00000002'.    I-STREAM 2
G02IS   ENTIT 1-2-X'00000012'.    PRIME GOA ORD NUM (#18)
G02CID  ENTIT 1-3-X'00000001'.    CPC 1
G02IID  ENTIT 1-3-X'00000001'.    I-STREAM 1
G02IS   ENTIT 1-3-X'00000020'.    PRIME GOA ORD NUM (#32)
G02CID  ENTIT 1-4-X'00000001'.    CPC 1
G02IID  ENTIT 1-4-X'00000002'.    I-STREAM 2
G02IS   ENTIT 1-4-X'00000027'.    PRIME GOA ORD NUM (#39)
GEND

```

Schematically, the super GOA is laid out as Figure 8 on page 302 shows.

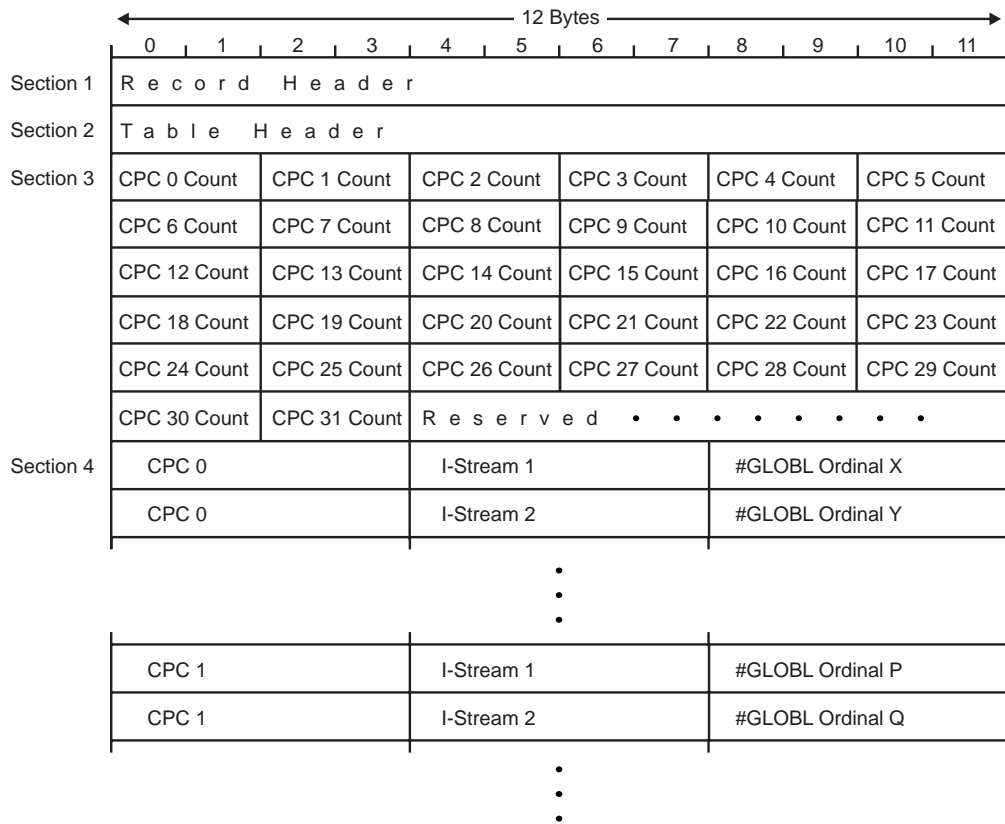


Figure 8. Super GOA Layout

Load Mode

In the header of each GOA record are 2 fields that determine whether the records listed in the GOA record should be loaded. These fields are GO1NUM and GO1CHN. The number of entries in the GOA list of records is in the GO1ENT field.

GO1NUM is a 1-byte field. The number in this byte is called a load mode. When GOGO retrieves this GOA, it compares the contents of GO1NUM to a byte in the subsystem user table entry for this subsystem user. If the values are equal, the records in the list are loaded. How the field in the subsystem user table is initialized is in "SIP for Globals" on page 261.

Once GOGO is finished loading the list of records defined in a GOA, it checks the contents of the GO1CHN field. If the first bit of the GO1CHN byte is 0, this shows to GOGO that there are no more GOA records to retrieve for this load mode. If the first bit of GO1CHN is 1, GOGO will continue to retrieve the next GOA record, the ordinal number of which is found in GO1FCH.

Figure 9 is an STC input that shows the use of load modes.

```

*      PRIME GOA RECORD
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)1.          #GLOBL ORDINAL NUMBER 1
*                                     STANDARD HEADER
GO1BID ENT  GO.                RECORD ID C'GO'
GO1FCH ENT  X'00000002'.        FORWARD CHAIN TO ORDINAL # 2
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0022'.        34 ENTRIES IN THIS TABLE
GO1NUM ENTIT 1-1-X'07'.        LOAD MODE X'07'
GO1CHN ENTIT 1-1-X'80'.        THERE IS A FORWARD CHAIN FOR X'07'
*                                     TABLE ENTRIES
.
.
.
GEND

*      OVERFLOW GOA RECORD
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)2.          #GLOBL ORDINAL NUMBER 2
*                                     STANDARD HEADER
GO1BID ENT  GO.                RECORD ID C'GO'
GO1FCH ENT  X'00000004'.        FORWARD CHAIN TO ORDINAL # 4
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0010'.        16 ENTRIES IN THIS TABLE
GO1NUM ENTIT 1-1-X'07'.        LOAD MODE X'07'
GO1CHN ENTIT 1-1-X'00'.        THERE IS NO FORWARD CHAIN FOR X'07'
*                                     TABLE ENTRIES
.
.
.
GEND

*      PRIME GOA RECORD
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)4.          #GLOBL ORDINAL NUMBER 4
*                                     STANDARD HEADER
GO1BID ENT  GO.                RECORD ID C'GO'
GO1FCH ENT  X'00000000'.        LAST GOA IN CHAIN
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0019'.        25 ENTRIES IN THIS TABLE
GO1NUM ENTIT 1-1-X'23'.        LOAD MODE X'23'
GO1CHN ENTIT 1-1-X'00'.        THERE IS NO FORWARD CHAIN FOR X'23'
*                                     TABLE ENTRIES
.
.
.
GEND

```

Figure 9. STC Input Using Load Modules

In Figure 9, if load mode X'07' is loaded, GOGO loads the records listed in #GLOBL 1 and #GLOBL 2, because their GO1NUM fields contain a X'07'. However, GOGO does not continue chasing the forward chain after #GLOBL 2, because the GO1CHN field contains 0.

When loading load mode X'23', GOGO retrieves #GLOBL1 and 2 but does not load any of the records listed because GO1NUM shows the wrong load mode. The GO1CHN field in #GLOBL 2 is ignored by GOGO because it shows the end of the GOA chain for load mode X'07'. GOGO continues to retrieve #GLOBL 4 and loads the records listed, because GO1NUM in that record contains a X'23'.

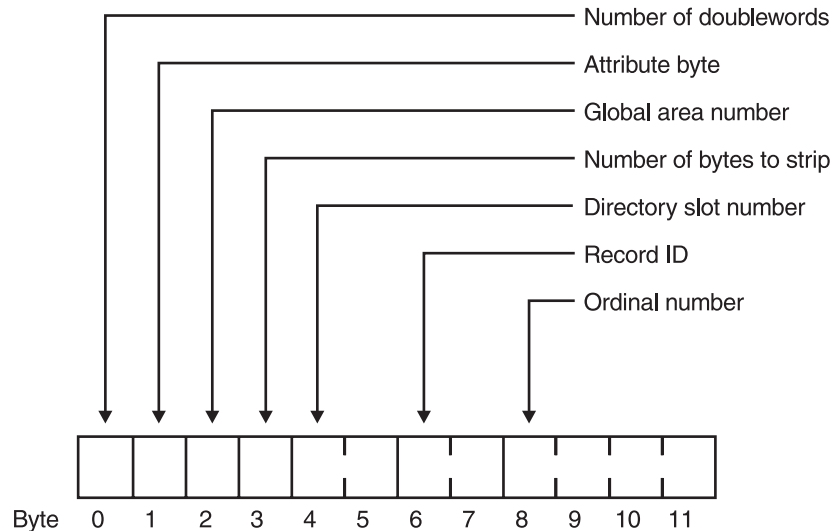
GOA Chain

GOGO first loads records into global area 1 (GL1), then into global area 2 (GL2), and then into global area 3 (GL3). Therefore, the list of records in an I-stream's GOA chain must be coded in this sequence.

The global blocks must be the first records loaded into the GL1 and GL3 areas, and, therefore, they must be the first GL1 and GL3 entries in the GOA list of records to load. The order that the global block entries appear must match the order that they are mapped by a GLOBZ macro call.

GOA List Entry

Each entry in a GOA list is 12 bytes long. Each set of 12 bytes contains information about one global record:



Number of Doublewords (Byte 0): Specifies the hexadecimal number of doublewords that will be loaded from the #GLOBL record for this global area.

When a global block is defined, this value must match exactly the number of doublewords declared for that global block by a GLOBZ call. For example, if @GLOBF declares 17 doublewords of storage, byte 0 for global block GL0BF must be X'11'.

Values range from X'01' (8 bytes) to X'84' (1056 bytes). Loading the global blocks in 8-byte increments can increase the block length to 1056 bytes in main storage. However, file restrictions limit the meaningful data in a block to 1055 bytes, including an 8-byte header.

Attribute byte (Byte 1): Tells the system what type of global is being loaded. The bit settings correspond to the following attributes:

- X'20' I-stream shared
- X'08' Extended global area resident
- X'04' Keypointable
- X'02' Subsystem user common
- X'01' No directory entry for this record.

Note: If the I-stream shared indicator is on in a GOA entry for an I-stream other than the main I-stream, the global load generates a system error, and restart processing ends.

Global Area Number (Byte 2): Shows the global area into which the global record is to be loaded as a number from 0 to 2, where 0 is GL1, 1 is GL2, and 2 is GL3.

Number of Bytes to Strip (Byte 3): Serves 2 purposes:

- This field can be used to enable loading from any designated point in the record so loading the header can be avoided. For records that are used as static data areas (tables, for example) and that are never filed, this prevents wasting the space occupied by the header once the record is read into main storage.

Note: Records with headers stripped are not keypointable and should not be defined as such. Conversely, records designated as keypointable must have 0 in byte 3.

- This field can also be used to concatenate several DASD records into one contiguous main storage block. When concatenating records, this field points to the byte that must immediately follow the previously loaded record. Standard values for this field are X'00', X'08', or X'10'. See "Record Concatenation".

Directory Slot Number (Bytes 4 — 5): Specifies the number from, 1 to 124, of the directory slot. There are 56 slots in the GL1 directory and 68 in the GL3 directory. Number 1 is the first slot for GL1; number 57 (coded X'39') is the first slot of the directory for GL3.

A 0 is coded for the slot number if the X'01' bit is set in the attribute byte (byte 1) of the GOA entry. See Record Concatenation.

Record ID (Bytes 6 — 7): Defines the record ID found in the header of the record to be loaded from DASD. The first 2 bytes of the global record to be loaded must be the same as the record ID coded in its GOA entry.

Ordinal number (Bytes 8 — 11): The #GLOBL ordinal number, in hexadecimal, of the record to be loaded into the slot indicated.

Considerations for Preparing Input

This section contains considerations for preparing the input.

Record Concatenation

The maximum number of bytes that can be loaded with one GOA entry is 1056 (X'84' doublewords). To create a larger block of data (say a large table), use record concatenation, which concatenates several global records to generate a single large block pointed to by a single directory slot. Concatenated global blocks can be loaded into any global area.

To create a concatenated block, you must code the pilot tape as follows. The first record in the block is loaded, with a slot number. Each following record is then loaded into the same global area, but with a 0 slot number in bytes 4 and 5 of the GOA entry and with the number of bytes to be stripped off in byte 3 to remove the header. All records in the block must be coded sequentially with no other records in between.

The following example of a series of entries in a GOA record show how to code STC input to generate a concatenated global block:

```
*   START OF CONCATENATED GLOBAL BLOCK
GO1CON  ENTIT 1-5-X'83000200'.    APPL RECOVERY CORE TABLE IN GL3
GO1EIX  ENTIT 1-5-X'0051'.        SLOT # 25 IN GL3
GO1EID  ENTIT 1-5-AR.             RECORD ID AR
GO1EON  ENTIT 1-5-X'00000037'.    LOAD FROM ORDINAL # 55
GO1CON  ENTIT 1-6-X'81010210'.    LOAD 81D MORE TO GL3
GO1EIX  ENTIT 1-6-X'0000'.        WITHOUT A SLOT NUMBER
GO1EID  ENTIT 1-6-AR.             RECORD ID AR
GO1EON  ENTIT 1-6-X'00000038'.    FROM ORDINAL # 56, SKIP 16 BYTES
```

```

GO1CON  ENTIT 1-7-X'81010210'.    LOAD 81D MORE TO GL3
GO1EIX  ENTIT 1-7-X'0000'.        WITHOUT A SLOT NUMBER
GO1EID  ENTIT 1-7-AR.            RECORD ID AR
GO1EON  ENTIT 1-7-X'00000039'.    FROM ORDINAL # 57, SKIP 16 BYTES
GO1CON  ENTIT 1-8-X'81010210'.    LOAD 81D MORE TO GL3
GO1EIX  ENTIT 1-8-X'0000'.        WITHOUT A SLOT NUMBER
GO1EID  ENTIT 1-8-AR.            RECORD ID AR
GO1EON  ENTIT 1-8-X'0000003A'.    FROM ORDINAL # 58, SKIP 16 BYTES
*      END OF CONCATENATED GLOBAL BLOCK
GO1CON  ENTIT 1-9-X'64000200'.    LOAD 64D TO GL3
GO1EIX  ENTIT 1-9-X'0042'.        SLOT NUMBER 10
GO1EID  ENTIT 1-9-TR.            RECORD ID TR
GO1EON  ENTIT 1-9-X'00000022'.    FROM ORDINAL # 34

```

Keypointable Global Records

Only the first 48 slots of the GL1 directory and the first 64 slots of the GL3 directory can be defined as keypointable. A record in one of these slots is keypointable if and only if:

- The X'04' bit of the global record's indicator byte (GOA entry, byte 1) is set on the pilot tape.
- The bytes-to-strip field (GOA entry, byte 3) is 0.
- The tag for the global record is defined as keypointable in either the KEYUC (for GL1) or the GLOUC (for GL3) macro.

Subsystem User Common Globals

To designate a global record as subsystem user common, the X'02' bit in the indicator byte of the record's GOA entry must be set on. This bit must be off for globals loaded into GL1.

I-Stream Shared Globals

To designate a global record as I-stream shared, the X'20' bit in the indicator byte of the record's GOA entry must be set on. This bit must be off for globals on I-streams other than the main I-stream.

Synchronization of Globals

Global synchronization is a process by which global fields and records are dynamically maintained among 2 or more active I-streams in a loosely coupled (LC) or tightly coupled (TC) system. Synchronization involves offline data record generation, online restart processing, and real-time application interface (SYNCC macro) processing.

The offline portion of global synchronization involves creation of the system interprocessor global table (SIGT). An SIGT is generated for each subsystem in each CPC. The GLSYNC macro is used to create the offline SIGT. SIGT also requires that space be reserved on DASD for the #SGFRI record type. For each subsystem user, one #SGFRI record is required for each global field that is to be synchronized. When no global fields require synchronization, at least 2 #SGFRI-type records are still needed, because 2 system globals are automatically synchronized by SIP. The #SGFRI record type is defined by coding the SIP RAMFIL macro. (See "Coding the SIP Macros" in *TPF System Generation* for more information on the GLSYNC and RAMFIL macros.)

During system restart, online initialization (completion of the process started offline) is performed. Segments CNPR, CNPS, and CNPT complete initialization of sections 0 and 1 of the SIGT. The main storage and file address of each synchronizable global field or record is calculated and moved into the SIGT. A minimum of at least one field, a dummy field, is defined as synchronizable.

Application programs initiate the online synchronization process when the SYNCC macro is issued. Copy member CSYN of the CSECT CCNUCL (Global Synchronization Service Routines) determines the type of SYNCC macro request and performs the requested function. Segment CNPU is activated to synchronize a field or record among all of the active I-streams in all active CPCs.

Requirements for Synchronization

The following requirements must be satisfied before global fields or records can be synchronized between tightly coupled or loosely coupled processors:

- The fields and records eligible for synchronization must be defined offline before system generation, and the control tables necessary to regulate the synchronization process must be constructed. Operating offline, SIP builds sections 0 and 1 of the SIGT (data macro SI0GT).
- A fixed file record called the synchronizable global field record (SGFR) is used to refresh the main storage copy of each synchronizable global field in the system. For each global field that a subsystem user seeks to synchronize, one SGFR is required. The SIGT that was built offline by SIP is initialized by program CNPR with the proper main storage and file address of each synchronizable global field and record.
- The global records must contain standard headers. The technique of removing headers and packing data into main storage (see "Record Concatenation" on page 305) is not supported for synchronizable global records.
- The programs that use the fields and records, particularly those that update the data, must be updated to incorporate the synchronization macro, SYNCC.
- Each program that depends on the most current value of the data in a global field or global record must, itself, ensure the presence of the most current data in main storage by issuing the SYNCC macro.

Note: Since the use of SYNCC in an LC environment is costly in terms of DASD overhead, it may be more useful to redefine the use of a global than to synchronize it.

Maximum Values for Synchronized Fields and Records

The maximum field and record sizes are:

- The maximum global field size for a synchronized field is 256 bytes.
- The maximum global record size for a synchronized record is 1056 bytes.

The maximum number of fields or records are:

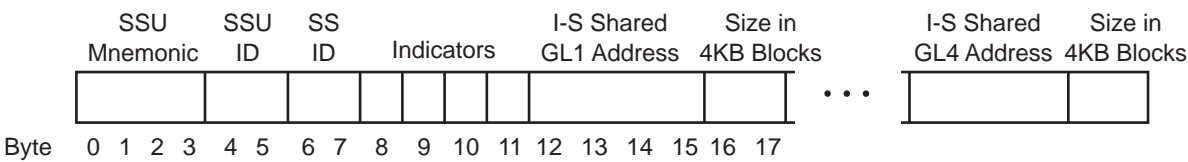
- The maximum number of synchronized global fields for a subsystem user is 256 fields.
- The maximum number of synchronized global records for a subsystem user is 256 records.

Locating Global Areas in a Dump

To locate the start of the global areas for each subsystem user on each I-stream, you must interrogate the subsystem user table (data macro MS0UT). In a dump, the subsystem user table (SSUT) is labeled with the tag SSU. An address for this tag is provided in the dump label index at the end of the dump.

The first 4 bytes of the SSUT contain the number of subsystem users in the system. The SSUT contains one entry for each SSU.

The length of each SSUT entry is defined by the label MU0LEN in DSECT MS0UT. A header occupies the first 12 bytes of each entry. The header is used to identify each entry with a subsystem user and is laid out as follows:



The rest of an entry contains the addresses to each I-stream’s unique primary and extended global areas as well as each I-stream’s global attribute tables, as Figure 10 on page 309 shows.

The formula for locating the address of any unique global area in a dump listing of the SSUT for any SSU/I-stream combination is:

$$\text{ADDRESS} = \text{START} + \text{S} + \text{G} + \text{I}$$

where

- START = address of the SSU tag + (MU0GLB - MU0CNT)
- S = SSU number × MU0LEN
- G = (global area number - 1) × MU0ASZ
- I = (I-stream number - 1) × MU0GSZ

These variables are defined in terms of the following symbolic labels found in MS0UT:

- MU0GLB - MU0CNT (length of standard header)
- MU0LEN (length of SSU entry)
- MU0ASZ (length of SSUT for a single global area)
- MU0GSZ (length of descriptor area for each I-stream).

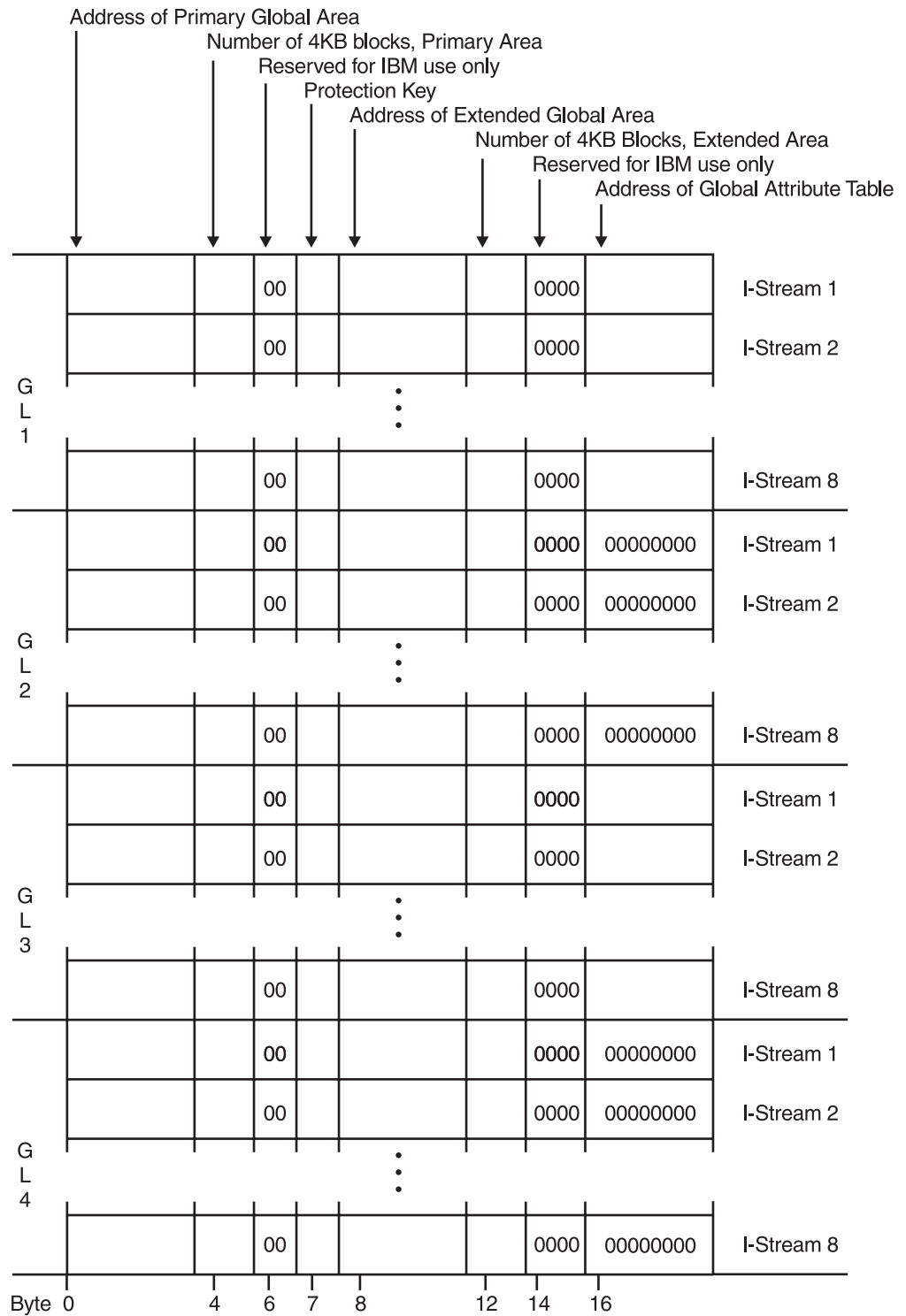


Figure 10. SSUT Entry

Examples of I-Stream Shared and Unique Globals

The following is STC input for the GOA chain for I-stream 1:

```

*      I-STREAM 1 PRIME GOA RECORD
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)1.          #GLOBL ORDINAL NUMBER 1
*                                     STANDARD HEADER
GO1BID ENT  GO.
GO1FCH ENT  X'00000002'.        FORWARD CHAIN AT ORDINAL # 2
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0002'.        2 ENTRIES THIS TABLE
GO1NUM ENTIT 1-1-X'07'.          LOAD MODE 07
GO1CHN ENTIT 1-1-X'80'.          THERE IS A FORWARD CHAIN
*                                     TABLE OF ENTRIES
*                                     RECORD 1
GO1CON ENTIT 1-2-X'84000000'.    LOAD 84D RECORD TO GL1
GO1EIX ENTIT 1-2-X'0009'.        SLOT # 9
GO1EID ENTIT 1-2-GL.             RECORD ID GL
GO1EON ENTIT 1-2-X'00000002'.    FROM ORDINAL # 2
*                                     RECORD 2
GO1CON ENTIT 1-3-X'84240000'.    LOAD 84D KEYPOINTABLE REC TO I-S shared GL1
GO1EIX ENTIT 1-3-X'000A'.        SLOT # 10
GO1EID ENTIT 1-3-GL.             RECORD ID GL
GO1EON ENTIT 1-3-X'00000003'.    FROM ORDINAL # 3
GEND
*      I-STREAM 1 FORWARD CHAIN GOA
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)2.          #GLOBL ORDINAL NUMBER 2
*                                     STANDARD HEADER
GO1BID ENT  GO.
GO1FCH ENT  X'00000000'.        NO FORWARD CHAIN
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0002'.        2 ENTRIES IN THIS TABLE
GO1NUM ENTIT 1-1-X'07'.          LOAD MODE 07
GO1CHN ENTIT 1-1-X'00'.          NO FORWARD CHAIN
*                                     TABLE ENTRIES
*                                     RECORD 3
GO1CON ENTIT 1-2-X'58020100'.    LOAD 58D SSU COMMON REC TO GL2
GO1EIX ENTIT 1-2-X'0079'.        GL3 SLOT # 65
GO1EID ENTIT 1-2-GL.             RECORD ID GL
GO1EON ENTIT 1-2-X'0000003E'.    FROM ORDINAL # 62
*                                     RECORD 4
GO1CON ENTIT 1-3-X'64060200'.    64D KYPTBL SSU COMMON REC TO GL3
GO1EIX ENTIT 1-3-X'0049'.        SLOT # 17
GO1EID ENTIT 1-3-GL.             RECORD ID GL
GO1EON ENTIT 1-3-X'0000003F'.    FROM ORDINAL # 63
GEND

```

The following is STC input for the GOA chain for I-stream 2:

```

*      I-STREAM 2 PRIME GOA RECORD
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)18.         #GLOBL ORDINAL NUMBER 18
*                                     STANDARD HEADER
GO1BID ENT  GO.
GO1FCH ENT  X'00000011'.        FORWARD CHAIN AT ORDINAL # 17
*                                     TABLE HEADER
GO1ENT ENTIT 1-1-X'0002'.        2 ENTRIES THIS TABLE
GO1NUM ENTIT 1-1-X'07'.          LOAD MODE 07
GO1CHN ENTIT 1-1-X'80'.          THERE IS A FORWARD CHAIN
*                                     TABLE OF ENTRIES
*                                     RECORD 1
GO1CON ENTIT 1-2-X'84000000'.    LOAD 84D RECORD TO GL1
GO1EIX ENTIT 1-2-X'0009'.        SLOT # 9
GO1EID ENTIT 1-2-GL.             RECORD ID GL
GO1EON ENTIT 1-2-X'00000002'.    FROM ORDINAL # 2
*                                     RECORD 2 I-STREAM SHARED
*

```

```

*      (Note that no record is defined here.
*      The corresponding record for I-stream 1
*      will appear in the directory slot for
*      for this I-stream.)
*
      GEND
*      I-STREAM 2 FORWARD CHAIN GOA
GO1GO  GSTAR 1.
BSTA06 ENT  (#GLOBL)17.          #GLOBL ORDINAL NUMBER 17
*
*      STANDARD HEADER
GO1BID ENT  GO.                  RECORD ID
GO1FCH ENT  X'00000000'.         NO FORWARD CHAIN
*
*      TABLE HEADER
GO1ENT ENTIT 1-1-X'0002'.        2 ENTRIES IN THIS TABLE
GO1NUM ENTIT 1-1-X'07'.         LOAD MODE 07
GO1CHN ENTIT 1-1-X'00'.         NO FORWARD CHAIN
*
*      TABLE ENTRIES
*
*      RECORD 3
GO1CON ENTIT 1-2-X'58020100'.    LOAD 58D SSU COMMON REC TO GL2
GO1EIX ENTIT 1-2-X'0079'.       GL3 SLOT # 65
GO1EID ENTIT 1-2-GL.            RECORD ID GL
GO1EON ENTIT 1-2-X'0000003E'.   FROM ORDINAL # 62
*
*      RECORD 4
GO1CON ENTIT 1-3-X'64060200'.    64D KYPTBL SSU COMMON REC TO GL3
GO1EIX ENTIT 1-3-X'0049'.       SLOT # 17
GO1EID ENTIT 1-3-GL.            RECORD ID GL
GO1EON ENTIT 1-3-X'00000040'.   FROM ORDINAL # 64
      GEND

```

The previous code produces a pilot tape that initializes the #GLOBL records so that 4 globals records are loaded.

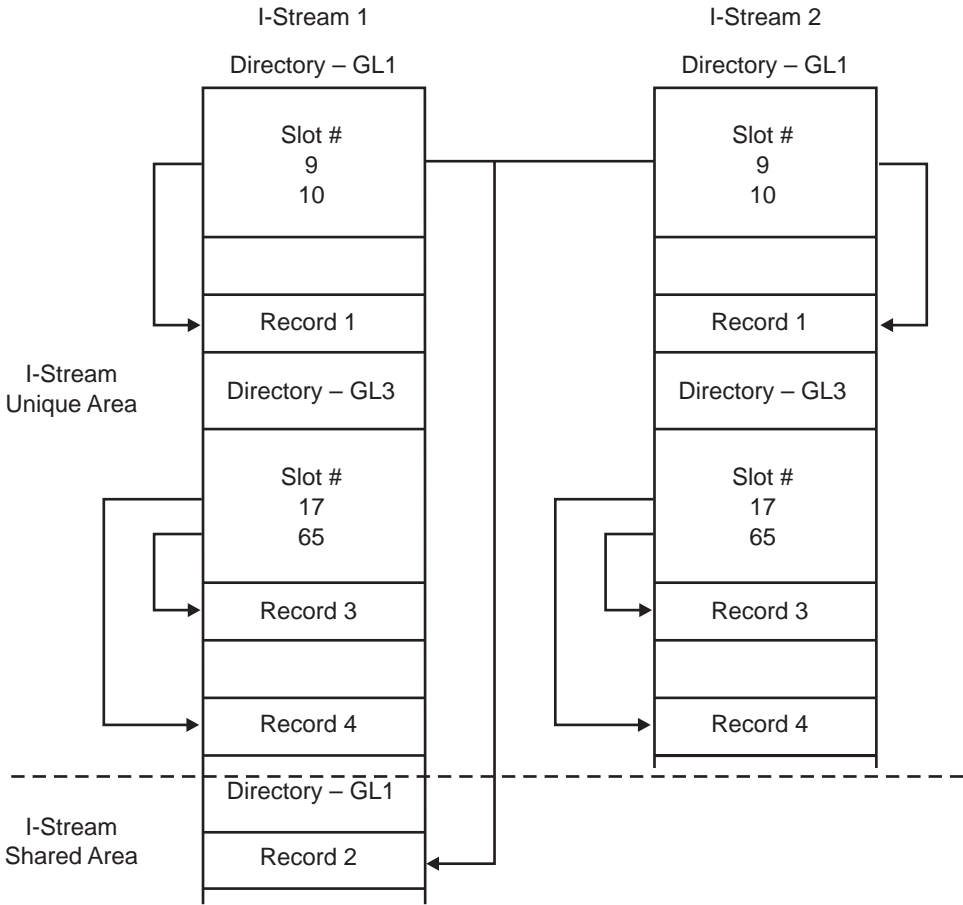
Records 1 and 3 are both I-stream unique since there is an item in each I-stream's GOA chain for both records. Each I-stream has its own main storage copy of both record 1 and record 3, but the file address for each global is shared. This is because the global is loaded to each I-stream from the same #GLOBL ordinal number.

Record 2 is an example of an I-stream shared global. Since there is an entry for record 2 only in I-stream 1's GOA chain, I-Stream 2's slot for record 2 (GL1 slot #10) contains the same values as I-stream 1's slot for record 2. Each I-stream points to the same main storage address and file address for record 2. This is what is meant by I-stream shared.

Note: If the I-stream shared indicator is on in a GOA entry for an I-stream other than the main I-stream, global load generates a system error, and restart processing ends.

Record 4, like records 1 and 3, is I-stream unique, but record 4 is loaded from a different #GLOBL ordinal number on each I-stream. Record 4 has not only a unique main storage copy for each I-stream but also a unique DASD copy for each I-stream.

The following diagram shows this relationship:



Main Storage Super GOA Copy

The following is an example of a main storage copy of a super GOA.

+000	G 00000	A C P F	+008	00000000	00300000	+010	00000000	00000000	+018	00000008	00100018
+020	00200028	00000000	+028	00000000	00000000	+030	00000000	00000001	+038	00000001	00000000
+040	00000002	00000010	+048	00000000	00000003	+050	00000010	00000000	+058	00000004	00000010
+060	00000000	00000005	+068	00000010	00000000	+070	00000006	00000010	+078	00000000	00000007
+080	00000010	00000000	+088	00000008	00000010	+090	00000001	00000001	+ABS	00000001	00000001
+0A0	00000002	00000010	+0A8	00000001	00000003	+0B0	00000010	00000001	+0B8	00000004	00000010
+0C0	00000001	00000005	+0C8	00000010	00000001	+0D0	00000006	00000010	+0D8	00000001	00000007
+0E0	00000010	00000001	+0E8	00000008	00000010	+0F0	00000002	00000001	+0F8	00000001	00000002
+100	00000002	00000010	+108	00000002	00000003	+110	00000010	00000002	+118	00000004	00000010
+120	00000002	00000005	+128	00000010	00000002	+130	00000006	00000010	+138	00000002	00000007
+140	00000010	00000002	+148	00000008	00000010	+150	00000003	00000001	+158	00000001	00000003
+160	00000002	00000010	+168	00000003	00000003	+170	00000010	00000003	+178	00000004	00000010
+180	00000003	00000005	+188	00000010	00000003	+190	00000006	00000010	+198	00000003	00000007
+1A0	00000010	00000003	+1A8	00000008	00000010	+1B0	00000004	00000001	+1B8	00000001	00000004
+1C0	00000002	00000010	+1C8	00000004	00000003	+1D0	00000010	00000004	+1D8	00000004	00000010
+1E0	00000004	00000005	+1E8	00000010	00000004	+1F0	00000006	00000010	+1F8	00000004	00000007
+200	00000010	00000004	+208	00000008	00000010	+210	00000005	00000001	+218	00000001	00000005
+220	00000002	00000010	+228	00000005	00000003	+230	00000010	00000005	+238	00000004	00000010
+240	00000005	00000005	+248	00000010	00000005	+250	00000006	00000010	+258	00000005	00000007
+260	00000010	00000005	+268	00000008	00000010	+270	00000000	00000000	+278	00000000	00000000
+280	00000000	00000000	+288	00000000	00000000	+290	00000000	00000000	+298	00000000	00000000
+2A0	00000000	00000000	+2A8	00000000	00000000	+2B0	00000000	00000000	+2B8	00000000	00000000
+2C0	00000000	00000000	+2C8	00000000	00000000	+2D0	00000000	00000000	+2D8	00000000	00000000
+2E0	00000000	00000000	+2E8	00000000	00000000	+2F0	00000000	00000000	+2F8	00000000	00000000
+300	00000000	00000000	+308	00000000	00000000	+310	00000000	00000000	+318	00000000	00000000
+320	00000000	00000000	+328	00000000	00000000	+330	00000000	00000000	+338	00000000	00000000
+340	00000000	00000000	+348	00000000	00000000	+350	00000000	00000000	+358	00000000	00000000
+360	00000000	00000000	+368	00000000	00000000	+370	00000000	00000000	+378	00000000	00000000
+380	00000000	00000000	+388	00000000	00000000	+390	00000000	00000000	+398	00000000	00000000
+3A0	00000000	00000000	+3A8	00000000	00000000	+3B0	00000000	00000000	+3B8	00000000	00000000
+3C0	00000000	00000000	+3C8	00000000	00000000	+3D0	00000000	00000000	+3D8	00000000	00000000
+3E0	00000000	00000000	+3E8	00000000	00000000	+3F0	00000000	00000000	+3F8	00000000	00000000
+400	00000000	00000000	+408	00000000	00000000	+410	00000000	00000000	+418	00000000	00000013

Figure 11. Example of the Super GOA

Main Storage Prime GOA Copy

The following is an example of a main storage copy of a prime GOA.

+000	G 00004	A C P F	+008	00000002	001A0700	+010	ABS00000	00000000	+018	84000000	0009 G L
+020	00000003	84040000	+028	000A G L	00000004	+030	09000000	0022 G L	+038	00000023	08040000
+040	0023 G L	00000024	+048	11000000	0024 G L	+050	00000025	03040000	+058	0025 G L	00000026
+060	64040000	0001 M E	+068	00000043	32040000	+070	0002 R T	00000044	+078	10040000	0003 P M
+080	00000041	10040000	+088	0004 G C	00000042	+090	84040000	002E Q X	+ABS	0000004B	64040000
+0A0	0005 T R	00000046	+0A8	64060000	0007 T R	+0B0	00000048	64040000	+0B8	002B T R	0000002
+0C0	64060000	002D T R	+0C8	0000002B	81000100	+0D0	0013 X U	0000001F	+0D8	09000100	0014 X U
+0E0	00000020	12000100	+0E8	0016 X C	00000018	+0F0	64040100	0006 T R	+0F8	00000047	64060100
+100	0008 T R	00000049	+108	64040100	002C T R	+110	00000028	64060100	+118	002F T R	0000002C
+120	64060100	0056 T R	+128	00000035	64040100	+130	0058 T R	0000002A	+138	64040100	0054 T R
+140	00000033	64060100	+148	005A T R	0000002E	+150	00000000	00000000	+158	00000000	00000000
+160	00000000	00000000	+168	00000000	00000000	+170	00000000	00000000	+178	00000000	00000000
+180	00000000	00000000	+188	00000000	00000000	+190	00000000	00000000	+198	00000000	00000000
+1A0	00000000	00000000	+1A8	00000000	00000000	+1B0	00000000	00000000	+1B8	00000000	00000000
+1C0	00000000	00000000	+1C8	00000000	00000000	+1D0	00000000	00000000	+1D8	00000000	00000000
+1E0	00000000	00000000	+1E8	00000000	00000000	+1F0	00000000	00000000	+1F8	00000000	00000000
+200	00000000	00000000	+208	00000000	00000000	+210	00000000	00000000	+218	00000000	00000000
+220	00000000	00000000	+228	00000000	00000000	+230	00000000	00000000	+238	00000000	00000000
+240	00000000	00000000	+248	00000000	00000000	+250	00000000	00000000	+258	00000000	00000000
+260	00000000	00000000	+268	00000000	00000000	+270	00000000	00000000	+278	00000000	00000000
+280	00000000	00000000	+288	00000000	00000000	+290	00000000	00000000	+298	00000000	00000000
+2A0	00000000	00000000	+2A8	00000000	00000000	+2B0	00000000	00000000	+2B8	00000000	00000000
+2C0	00000000	00000000	+2C8	00000000	00000000	+2D0	00000000	00000000	+2D8	00000000	00000000
+2E0	00000000	00000000	+2E8	00000000	00000000	+2F0	00000000	00000000	+2F8	00000000	00000000
+300	00000000	00000000	+308	00000000	00000000	+310	00000000	00000000	+318	00000000	00000000
+320	00000000	00000000	+328	00000000	00000000	+330	00000000	00000000	+338	00000000	00000000
+340	00000000	00000000	+348	00000000	00000000	+350	00000000	00000000	+358	00000000	00000000
+360	00000000	00000000	+368	00000000	00000000	+370	00000000	00000000	+378	00000000	00000000
+380	00000000	00000000	+388	00000000	00000000	+390	00000000	00000000	+398	00000000	00000000
+3A0	00000000	00000000	+3A8	00000000	00000000	+3B0	00000000	00000			

Sample STC Card Images for Global Block Creation

@NC1NC	ENT	(#MISCS)7.		00270000
@NS1NS	ENT	(#MISCS)8.		00280000
@P9ADR	ENT	(#MISCS)10.		00290000
@P9PDC	ENT	X'00000021'.	33	00300000
@Q6PI1	ENT	(#MISCS)0.		00310000
@Q6PI2	ENT	(#MISCS)1.		00320000
@Q6PI3	ENT	(#MISCS)2.		00330000
@ROTTY	ENT	(#MISCL)7.		00340000
@STTSF	ENT	(#MISCS)5.		00350000
@TTTEM	ENT	(#MISCL)0.		00360000
@UITFT	ENT	(#MISCS)9.		00370000
@U7CRC	ENT	(#QCRR)1.		00380000
@ALPHA	ENT	X000060.	6	00390000
@CRCOD	ENT	RC.		00400000
@HAALC	ENT	CO.		00410000
@LSTCH	ENT	X'0000'.		00420000
@MAXBK	ENT	X'0156'.	342	00430000
@MAXCL	ENT	X'0040'.	64	00440000
@MAXHL	ENT	X'0045'.	69	00450000
@MINBK	ENT	X'0105'.	261	00460000
@NBADY	ENT	X000070.		00470000
@NOCUR	ENT	X000080.	8	00480000
@NOREM	ENT	X0014D0.	333	00490000
@NRDET	ENT	X0000A0.	10	00500000
@NRGRO	ENT	X0016E0.	366	00510000
@NRRDI	ENT	X0003D0.		00520000
@OADET	ENT	X000050.	5	00530000
@OAGRO	ENT	X0014F0.	335	00540000
@OMSGI	ENT	X'0005'.	5	00550000
@P9PDO	ENT	X'000B'.		00560000
@P9PNL	ENT	X'0004'.	4	00570000
@XIPCT	ENT	X000040.	TON'S 0-3 NOT ASSIGNED XILD	00580000
@XLSLC	ENT	X000100.	# LOW SPEED LINES,16 LOW SPEE LINES 00-0F	00590000
@XLSLQ	ENT	X0000A0.	# LS ENTRIES(XTQC)	00600000
@XLSTC	ENT	X0000E0.	# OF L/S TERMINALS	00610000
@XMXML	ENT	X0000A0.	MAX MESSAGE LENGTH IN SEGMENTS	00620000
@XMXRC	ENT	X000050.	MAX RETRIEVAL COUNT	
@XMXRT	ENT	X0000F0.	MAX RETRIEVAL TIME LIMIT -15 MIN	00640000
@XNHST	ENT	X'0016'.	# H/S M/S TERMINALS	00650000
@XSATS	ENT	X'0007'.	# ENTRIES PER XSAT RECORD	00660000
@XSYLC	ENT	X000130.	# ENTRIES IN XOCT	00670000
@XSYTC	ENT	X'0028'.	# ENTRIES IN XTRT	00680000
@XTATS	ENT	X0000A0.	# ENTRIES PER XTAT RECORD	00690000
@X1SDL	ENT	X'0048'.	72 SDL'S PER XDLS	00700000
@TTHAM	ENT	TSTAL.		00710000
@CRAS	ENT	X'010000'.		00720000
@CRCCC	ENT	LAX.		00730000
@FCOCC	ENT	LAX.		00740000
@PARSC	ENT	LAX.		00750000
@CRCORD	ENT	X'01'.		00760000
@PQ5NR	ENT	X0010.		00770000
@XIPLG	ENT	X'FF'.	INPUT LOGGING ON	00780000
@XSYST	ENT	X0030.	RES EOM/S- H/S AND L/S	00790000
	GEND			00800000
SDMU	ENTER	GL0BCLAB0001		00810000
GL0BC	GSTAR	CREATE,1.		00820000
BSTA06	ENT	(#GL0BL)4.		00830000
0	ENT	X0C7D300200.		00840000
@U1CAL	ENT	X01EE2C5D700F10.	SEP	00850000
		X01FD6C3E3010F0.	OCT	00860000
		X01ED5D6E5012E0.	NOV	00870000
		X01FC4C5C3014C0.	DEC	00880000
		X01FD1C1D5016B0.	JAN	00890000
		X01CC6C5C2018A0.	FEB	00900000
		X01FD4C1D901A60.	MAR	00910000
		X01EC1D7D901C50.	APR	00920000
		X01FD4C1E801E30.	MAY	00930000

		X@1ED1E4D50202@.	JUN	;00940000
		X@1FD1E4D30220@.	JUL	;00950000
		X@1FC1E4C7023F@.	AUG	;00960000
		X'1EE2C5D7025E'.	SEP	;00970000
		X'1FD6C3E3027C'.	OCT	0ABS0000
@UIGMT	ENT	2000.		00990000
@UITYM	ENT	1200.		01000000
@UIMID	ENT	X@000001E0@.	480	01010000
@COCLO	ENT	X@000004B0@.	1200	01020000
@UIZID	ENT	X'00000000'.		01030000
@IFLTN	ENT	X@00000000000000@.		01040000
@CFLTNT	ENT	0000.		01050000
@DFLTNT	ENT	0000.		01060000
@PFLTNT	ENT	0000.		01070000
@GMTDA	ENT	X'0120'.	288	01080000
@NARAV	ENT	X'0000'.		01090000
@NRDDP	ENT	X@0007@.		01100000
@NRFMA	ENT	X@0000@.		01110000
@NRGDP	ENT	X'011F'.	287	01120000
@N9ARS	ENT	X'011F'.	287	01130000
@C0DEF	ENT	X'00'.		01140000
@UIDAY	ENT	X'0120'.	288	01150000
@UIDMO	ENT	X'011F'.	287	01160000
@NBADT	ENT	X@0000C9@.		01170000
@TTHDM	ENT	120017OCT.		01180000
@GMTDY	ENT	17OCT.		01190000
@C0TOL	ENT	X'00'.		01200000
@UIDMT	ENT	17OCT.		01210000
@VCHK	ENT	X'00'.		01220000
@VSTAT	ENT	I.		01230000
	GEND			01240000
SDMU	ENTER	GL0BD0000001,TPF DEV,70131		01250000
GL0BD	GSTAR	1.		01260000
BSTA06	ENT	(#GL0BL)35.		01270000
0	ENT	X@C7D303@.		01280000
@FQMIN	ENT	X'03E8'.		01290000
@FTADR	ENT	X'000000000000000'.		01300000
@FQADJ	ENT	X'0000'.		01310000
@FQNC	ENT	X'0096'.		01320000
@TMSLR	ENT	X'00000101'.		01330000
@FQCTX	ENT	X'0008'.		01340000
@ISNAM	ENT	TPF AIR.		01350000
	GEND			01360000
SDMU	ENTER	GL0BE0000001,TPF DEV,70131		01370000
GL0BE	GSTAR	1.		01380000
BSTA06	ENT	(#GL0BL)36.		01390000
0	ENT	X@C7D30304@.		01400000
@F1KEY	ENT	X'0000'.		01410000
@TMACT	ENT	X'00'.		01420000
@HIORD	ENT	X'0000'.		01430000
@TKTNO	ENT	X'00000000'.		01440000
@COIBM	ENT			01450000
	GEND			01460000
SDMU	ENTER	GL0BF0000001,TPF DEV,70131		01470000
GL0BF	GSTAR	1.		01480000
BSTA06	ENT	(#GL0BL)37.		01490000
0	ENT	X@C7D303@.		01500000
	GEND			01510000
SDMU	ENTER	GL0BG0000001,TPF DEV,70131		01520000
GL0BG	GSTAR	1.		01530000
BSTA06	ENT	(#GL0BL)38.		01540000
0	ENT	X@C7D30304@.		01550000
	GEND			01560000
SDMU	ENTER	GL0BP0000001		01570000
GL0BP	GSTAR	1.		01580000
BSTA06	ENT	(#GL0BL)62.		01590000
0	ENT	X'C7D3'.		01600000

	GEND	01610000
SDMU	ENTER GLOBQ00000001	01620000
GLOBQ	GSTAR 1.	01630000
BSTA06	ENT (#GLOBL)63.	01640000
0	ENT X@C7D3@.	01650000

Examples of Coding the SYNCC Macro

The following coding examples are illustrations, not the definitive method for using the SYNCC macro; they should be consulted only as examples. *TPF General Macros* contains a detailed explanation of the SYNCC macro.

Updating a Single Synchronized Field

Updating a single synchronizable field requires the insertion of the SYNCC macro at 2 points in the logic. The synchronization macro usually replaces the GLMOD and FILKW macros.

In the following example, the field @AFIELD is *locked* (statement 7), and the main storage copy is *refreshed* from the file copy. The protection key is set to the value appropriate for the field, and control is returned to the problem program at the next sequential instruction. A *hold* is maintained on the file copy of the record for the field by the processor. The request to synchronize the field between all processors is issued in statement 14. The protection key is returned appropriately for the problem program. The synchronization process for the other active processors is initiated by filing the synchronization copy of the field, sending messages to the other active processors, and releasing the *hold* on the synchronization record.

```

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
1)   .
2)   .
3)   LABEL    GLOBZ REGR=R1
4)   .
5)   .
6)   .
7)   LABEL1   SYNCC LOCK,@AFIELD
8)   .      **
9)   .      ** Logic may not modify working storage      **
10)  .      ** without the proper use of GLMOD and        **
11)  .      ** FILKW macros.                               **
12)  .      **
13)   ST      RG5,@AFIELD      Update the single field
14)  LABEL2   SYNCC SYNC,@AFIELD
15)  .
16)  .

```

Updating Multiple Synchronized Fields

Requests to update multiple synchronizable fields must be done serially. Inserting a SYNCC macro before and after each field update request will substantially increase the size of a program. To avoid issuing many SYNCC macros, the database design should be reevaluated in an LC system. GLMOD and FILKW macros may have to remain in the program to properly update nonsynchronizable fields or records. Two synchronized fields cannot be updated with a single SYNCC macro call.

In the following example, the field @AFIELD is *locked* (statement 6), and the main storage copy is *refreshed* from the file copy. The protection key is set to the value appropriate for the field, and control is returned to the problem program at the next sequential instruction. A *hold* is maintained on the file copy of the record for the field by the processor. The request to synchronize the field between all processors is issued in statement 14. The protection key is returned appropriately for the problem program. The synchronization process for the other active processors is initiated by

filing the synchronization copy of the field, sending messages to the other active processors and releasing the *hold* on the synchronization record. The entire process is repeated for @BFIELD.

```

1) |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
2) .
3) LABEL    GLOBZ REGR=R1
4) .
5) .
6) LABEL1    SYNCC LOCK,@AFIELD
7) .
8) .    **
9) .    ** Logic may not modify working storage **
10) .    ** without the proper use of GLMOD and **
11) .    ** FILKW macros. **
12) .    **
13) .    ST    RG5,@AFIELD    Update single field
14) LABEL2    SYNCC SYNC,@AFIELD
15) .
16) .    ** Additional logic **
17) .
18) LABEL3    SYNCC LOCK,@BFIELD
19) .    **
20) .    ** Logic may not modify working storage **
21) .    ** without the proper use of GLMOD and **
22) .    ** FILKW macros. **
23) .    **
24) .    ST    RG5,@BFIELD    Update single field
25) LABEL4    SYNCC SYNC,@BFIELD
26) .
27) .

```

Updating Nonsynchronized and Synchronized Fields

In this example, the field @AFIELD is *locked* (statement 6) and the main storage copy is *refreshed* from the file copy. The protection key is set to the value appropriate for the field and control is returned to the problem program at the next sequential instruction. A *hold* is maintained on the file copy of the record for the field by the processor. In following code (statements 7 through 12), the program properly addresses the field, modifies the data, and in this example, uses a store register to update the field (statement 13). The request to synchronize the field across all processors is then issued (statement 14) and the program continues. As a result of the request for synchronization (statement 14), the protection key is returned appropriately for the problem program and the synchronization process for the other active processors is initiated by filing the synchronization copy of the field, sending messages to the other active processors, and releasing the *hold* on the synchronization record. The update to @BFIELD is by conventional use of the GLMOD and FILKW macros.

```

1) |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
2) .
3) LABEL    GLOBZ REGR=R1
4) .
5) .
6) LABEL1    SYNCC LOCK,@AFIELD
7) .
8) .    **
9) .    ** Logic may not modify working storage **
10) .    ** without the proper use of GLMOD and **
11) .    ** FILKW macros. **
12) .    **
13) .    ST    RG5,@AFIELD    Update single field
14) LABEL2    SYNCC SYNC,@AFIELD
15) .

```

```

16) .    ** Additional logic                **
17) .
18) LABEL3  GLMOD
19) .
20) .    **                                **
21) .    ** Logic may not modify working storage **
22) .    ** without the proper use of GLMOD and  **
23) .    ** FILKW macros.                      **
24) .    **                                **
25) .      ST      RG5,@BFIELD      Update single field
26) LABEL4  FILKW R,@GBLCC
27) .
28) .

```

Loaders

A loader is a tool for introducing new or modified system components, programs, or data to a TPF system. All loaders have both an online and offline component.

Offline Component

Takes system components, programs, or data from the MVS data sets and writes them to a storage medium such as tape, DASD, virtual reader, or user-defined device.

Online Component

Takes the system components, programs, or data from the storage medium and loads them to their proper location on the online system.

The load functions of a TPF system are handled by the general file loader, data loader, E-type loader, and auxiliary loader. Only the general file loader is used for an initial full load (see “Loading System Components to a New TPF System” on page 322).

Once you complete a full load, you do not need to do another except for some DASD configuration or core image restart (CIMR) changes.

General File System Components

The programs and keypoints needed to start or restart the system, and other programs and keypoints, get stored in the general file. The following list shows the general file contents:

- IPLA and the volume serial number (VOLID)

Note: This copy of IPLA is used when the loader general file is IPLed.

- Keypoints from an MVS object library
- IPLA

Note: This copy of IPLA is loaded to online modules.

- IPLB
- Core Image Restart (CIMR) area components
 - Control program (CP)
 - Program Allocation Table (IPAT)
 - In-core dump formatter (ICDF) program
 - Online segment of the general file loader (ACPL)
 - Global synchronization table (SIGT)
 - Record identifier attribute table (RIAT)
 - File address compute program table (FCTB)
 - USR1
 - USR2
- E-Type Programs and Data Records
 - E-type programs from an MVS object library
 - Loader Control Record (LDCRL)
 - Program Version Records (PVRs)
- General File IPL Keypoints
- Online Module IPL Keypoints
- Online Keypoint Patches

There are 2 types of general files for HPO: one for the basic subsystem (BSS) and one for subsystems. Only the basic subsystem has the code necessary for an IPL.

Changing CIMR Components

A change to any CIMR component that causes the length of the component to increase by more than the buffer provided at system generation, requires the following actions:

1. Run the SIP program GTSZ to put the length of the changed component (including the buffer) in the SKGTSZ member of the SIP SYSRCE macro library.
2. Assemble CTKA and CTKX (both online and general file versions) to calculate the size of the CIMR and the file addresses for the CIMR components.
3. Perform a full load.

Note: Any change in the allocation of a CIMR component requires a full load.

Loading System Components to a New TPF System

The general file loader is used to load all system components to a new TPF system or to selectively update any system component. The general file loader is only necessary at system generation or in an emergency load condition in which no fallback image exists. The general file loader can only be used to load to fixed image 1.

Note: You must stop the entire online system to use the general file loader. Therefore, if you are loading system components to an existing TPF system, it may be advantageous to use the auxiliary loader. You do not have to stop the system to use the auxiliary loader. See “Loading System Components to an Existing TPF System” on page 341 for more information.

After you assemble all system and application programs and generate their data records, do the following to load the system components to a new TPF system:

1. Initialize the general file.
2. Format the general file and online modules.
3. Create a general file load deck.
4. Load system components to the general file.
5. IPL the general file.
6. Load fixed-file records.
7. IPL the online module.

Initializing the General File

You must initialize and format the general file before you can load system components to it. The system initialization program (SIP) does the initialization. SIP uses the MVS utility program ICKDSF to load IPLA and the volume serial number (VOLID) to the general file.

Formatting the General File and Online Modules

The real-time disk formatter prepares the general file and online modules to receive system components.

Creating a General File Load Deck

This section shows the steps you must take to create a load deck (assuming SIP has been run successfully).

Enter the JCL Cards

The following JCL cards are used to run the general file loader offline segment (ALDR); *nn* is the release identification and *ssid* is the subsystem ID.

```
//ALDRGF JOB (82F91,7323E),'SIP ACP ',
// MSGLEVEL=1,CLASS=F,
// MSGCLASS=A,TIME=15
//LOAD40 EXEC PGM=TPFLDRnn,REGION=9000K,PARM='ALDR,CLMSIZE=4000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
// DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
// DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SALTB DD DSN=ACP.SALTB.LINK.RELnn.ssid,DISP=SHR
//GENFIL DD DSN=GNFL.ssid,DISP=OLD,
// UNIT=3380,VOL=(PRIVATE,,,,SER=xxxxxx)
//GENFI2 DD DSN=GNF2BSS,DISP=OLD,
// UNIT=3380,VOL=(PRIVATE,,,,SER=xxxxxx)
//PRINT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//LDRTRACE DD SYSOUT=*
//ALDRCP DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//OBJDD DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//INPUT DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
/*
```

Figure 13. ALDR Run Load Deck Example

Notes:

1. DD names SALTB, OBJDD, GENFIL, PRINT, INPUT, and LOADMOD are required.
2. DD name LDRTRACE is required only if you specify TRACE(ON) as a parameter.
3. Data set concatenation is sequence-dependent.

TPFLDR EXEC Card: The TPFLDR EXEC card identifies the specified version of the TPF offline loader program (TPFLDR).

These are the parameters that can be specified on the EXEC card.

1. ALDR
indicates the general file loader (ALDR) is to be executed.
2. CLMSIZE = x
where x is greater than or equal to the MVS file size of the largest C load module to be loaded. If a C load module contains a higher than normal ratio of VCONs to executable code, CLMSIZE must be larger. ALDR searches only the OBJDD DSN (not LOADMOD DSN) when CLMSIZE = 0.
3. TRACE(ON|OFF)
If you specify TRACE(ON), selective trace data will be written to the LDRTRACE DD name. This trace data can be useful to system programmers to debug problems that occur while running TPFLDR. TRACE(ON) is primarily intended for TPF development personnel for this purpose. TRACE(OFF) is the default. If you specify TRACE(OFF), no trace data is written.

STEPLIB Card: The STEPLIB card identifies the partitioned data sets that contain the offline loader programs.

SALTB Card: The SALTB card identifies partitioned data sets that contain the system allocator table (SALTBL). The SALTBL contains information necessary for the TPF linkage editor to resolve virtual address constants (VCONs). These VCONs can be program addresses, tape names, macro parameters, and others that are put in the SALTBL to allow relocation of resources in the online system. See “System Allocator” on page 399 for more information on the SALTBL.

GENFIL Card: The GENFIL card identifies the general file to which offline data is written.

GENFI2 Card: The GENFI2 card identifies the general file DSN for the 4KB region to which offline programs and data are written.

ADATATMP Card: The ADATATMP card specifies a temporary data set used to hold ADATA files before they are written to the output medium.

PRINT Card: The PRINT card causes an output listing that details all system components loaded to the general file to be produced. Errors that do not cause the program to abend are also written in this listing. Abend error messages and informational messages are sent to the operator console. For more information about error messages, see *Messages (System Error and Offline)* and *Messages (Online)*.

ALDRCP Card: The ALDRCP card identifies the partitioned data set that contains the link-edited control program and FACE table. The FACE table and control program are assembled and link-edited using standard MVS facilities. For more information, see *TPF System Generation*.

LOADMOD Card: The LOADMOD card identifies the partitioned data set (LOADMOD) that contains the link-edited C load modules. When an E-type program is being loaded, the data sets identified by the LOADMOD card are searched before the data sets identified by the OBJDD card are searched. The C load modules are compiled and link-edited using standard MVS facilities. For more information, see *TPF System Generation*.

OBJDD Card: The OBJDD card identifies object libraries. Object libraries are partitioned data sets of assembled or compiled TARGET(TPF) programs. All programs to be loaded must be in object libraries with the exception of the control program, the FACE table, and C load modules. Library contents are maintained and updated using MVS procedures. For more information see *TPF System Generation*.

SYSPRINT Card: The SYSPRINT card identifies the device to which output is printed. It should be included in the ALDR load deck or some error messages related to the C load module will be routed to the **stdout** device. See *TPF C/C++ Language Support User's Guide* for more information on stdout.

Enter the Input Control Cards

Figure 14 on page 325 shows a sample ALDR input control load deck. The example explains the order of the cards and states if the cards are required or optional.

```

=====
Initialization part of the load deck - Must be first input control cards
=====
LOADER  CC    40          <-- Required, must be first
LOADER  IMAGE CLEAR      <-- Optional, immediately after CC
LOADER  LOAD  CTKX 40     <-- Optional, (required after IMAGE CLEAR)
GF      CALL  KEYPTCTKAGF <-- Optional, last initialization statement
GF      CALL  KEYPTCTKVGf
=====

Main part of the load deck - Must be second input control cards
=====
LOADER  LOAD  CP    40     <-- Required, before file-resident programs
LOADER  LOAD  IPAT 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  KEYPT       <-- Optional, anywhere after LOAD CP
LOADER  CALL  KEYPTCTK040  <-- Optional, immediately after LOAD KEYPT
LOADER  CALL  KEYPTCTK140
LOADER  CALL  KEYPTCTK241
LOADER  CALL  KEYPTCTK340
LOADER  CALL  KEYPTCTK440
LOADER  CALL  KEYPTCTK540
LOADER  CALL  KEYPTCTK640
LOADER  CALL  KEYPTCTK940
LOADER  CALL  KEYPTCTKA40
LOADER  CALL  KEYPTCTKB40
LOADER  CALL  KEYPTCTKC40
LOADER  CALL  KEYPTCTKD40
LOADER  CALL  KEYPTCTKE40
LOADER  CALL  KEYPTCTKI40
LOADER  CALL  KEYPTCTKM40
LOADER  CALL  KEYPTCTKV40
LOADER  CALL  KEYPTCTKA40  C
LOADER  PROG-MOD-BASE CLEAR <-- Opt, after LOAD CP & before LOAD OPL
LOADER  ELDR CLEAR        <-- Optional, anywhere after LOAD CP
LOADER  LOAD  OPL         <-- Optional, anywhere after ELDR CLEAR
PARS40
PARS4D
PARSP1
LOADER  LOAD  AP          <-- Optional, anywhere after LOAD CP
LOADER  CALL  PROG COSY40 <-- Optional, immediately after LOAD AP
LOADER  CALL  PROG BXAX40
LOADER  CALL  PROG CCKP40
LOADER  CALL  PROG COHA40
LOADER  CALL  PROG COHB40
LOADER  CALL  PROG UF00P1
REP 000348 0014700 <-- Optional, immediately after CALL PROG -> UF00P1
LOADER  LOAD  ACPL 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  FCTB 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  ICDF 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  IPLA 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  IPLB 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  SIGT 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  RIAT 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  USR1 40     <-- Optional, anywhere after LOAD CP
LOADER  LOAD  USR2 40     <-- Optional, anywhere after LOAD CP
LDT
/*

```

Figure 14. ALDR Input Control Load Deck Example

Figure 15 on page 326 shows a sample ALDR input control load deck that makes use of the PATH card to load an FCTB in program object format from an HFS path.

```

=====
      Initialization part of the load deck - Must be first input control cards
=====
LOADER  CC      40              <-- Required, must be first
=====
      Main part of the load deck - Must be second input control cards
=====
LOADER  PATH  FCTBPATH          This search path can be used to find FCTB
      /u/tpf41/product_components/system/;                      X
      /u/tpf41/product_components/test/;                        X
      /u/user1/my_project/tables
LOADER  LOAD  FCTB 40/FCTBPATH <-- FCTBPATH indicates search path to use
      LDT              <-- Required, must be last
/*

```

Figure 15. ALDR Input Control Load Deck Example Showing FCTB in Program Object Format from HFS

CC Card:

```

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER  CC      uu  vv,vv,vv,vv,vv,vv,vv,vv,vv,vv

```

uu = SALTBL version
vv = Version number of operational program (PARS) list

The CC card contains control information for the program. It supplies the version number of the SALTBL to be used for link-editing and the version numbers of any operational program lists to be used to build the directory in main storage. As many as ten lists can be specified. The only card that can precede the CC card is a REP card for the loader. Otherwise the loader willabend.

Image Clear Card:

```

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER  IMAGE CLEAR

```

The Image Clear card ensures clean image definitions. If it is present, ALDR will set an indicator for ACPL to perform the initialization of various image-related records. Note that with a clear card IPLA, IPLB, CTKX, and all the CIMR components need to be loaded.

Attention: The image clear card will cause all image definitions to be cleared on the online system.

Comment Card:

```

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
*comments

```

Comment cards can be placed anywhere in the load deck. Comments are identified by an asterisk (*) in card column 1. As far as control card sequencing is concerned, comment cards are ignored in this section of this publication.

Patching the General File Loader Offline Segment:

```

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx  ALDR

```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

The general file loader offline segment first checks for a REP card. The REP card indicates that the offline segment is to be patched before it executes. Any control card other than a REP card will return the program to normal load mode. The REP card must contain ALDR in columns 75–78 or the run will terminate.

Load CTKX Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD CTKX vv

vv = version number

The Load CTKX card causes the image pointer record (CTKX) to be loaded to the general file in a 4KB record. A switch is set in the loader control record to indicate to the general file loader online segment that CTKX was loaded. When a new CTKX is loaded it is compared to the existing version on the GF. If the start ordinal of a CIMR or IPL component is different in the new CTKX, or the allocated size is too small, that component must also be loaded.

Patching: One or more REP cards can follow the Load CTKX card. Each REP card must contain CTKX in card columns 75–78 and the version number in columns 79–80.

Call General File Keypoint Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

GF CALL KEYPTnnnnvv

nnnn = Keypoint name
 vv = version number

The specified keypoints are written to the general file keypoint area. Only 2 keypoints contain information for the general file that is not the same as the information they carry for the online modules:

CTKA Keypoint A

CTKV Keypoint V

These keypoints have pertinent information about the general file and are used during the general file IPL. They are not used to configure the online system. These keypoints have to be called only if the online versions are called. This is because the Call Online Keypoint card will cause any keypoint not called by the Call General File Keypoint card to be written to the general file keypoint area and the online keypoint area. See “Call Online Keypoint Card” on page 329.

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnvv

z = Any character except *
 aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 nnnn = Associated keypoint name
 vv = version number

A keypoint is patched by placing one or more REP cards after its Call General File Keypoint card. Each REP card must contain the same name and version number in card columns 75–80 as that punched in columns 21–26 of the corresponding Call General File Keypoint card.

Load Control Program Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD CP vv

vv = version number

The offline loader builds a copy of the control program (CP) in main storage from the load module library. The CP is then filed on the general file as a chain of 4KB records (known as core-image records). A version number must be specified in columns 21–22. Switches are set in the loader control record to inform the general file loader online segment that the control program was loaded.

Because of main storage utilization, the CP must be loaded before file-resident programs are loaded to the general file. This restriction includes application program loads via OPL/PARS and LOADER LOAD AP cards. Attempts to load the CP after file-resident programs will cause the loader toabend.

Note: The Load Control Program Card is only allowed in BSS.

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx ccccccvv

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 ccccc = CP CSECT name
 vv = version number

Patch the control program by placing REP cards after the Load Control Program card. Control program patching is accomplished by patching CP CSECTS individually starting at relative address zero. Each REP card must contain the CP CSECT name starting in column 73.

Load IPAT Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD IPAT vv

vv = version number

The Load IPAT card causes the program allocation table (PAT) to be loaded to the general file as core-image records. The time stamp in the header of system allocator table (SALTBL) will be used to validate the IPAT. If the time stamp in the header of the IPAT does not match the time stamp of the system allocator table (SALTBL), the load will be aborted.

Patching: One or more REP cards can follow the Load IPAT card. Each REP card must contain the IPAT in card columns 75–78 and the version number in columns 79–80.

Load Keypoint Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LOADER LOAD KEYPT

The Load Keypoint card tells the offline segment that there are keypoints to be loaded to the working areas of the online modules. (The general file loader loads keypoints directly to the working keypoint area (#KEYPT), not the staging area (#KSAX)). The keypoints are identified by Call Online Keypoint cards that must follow the Load Keypoint card. These cards must come after the Call General File Keypoint card and the Load Control Program card.

Call Online Keypoint Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

GF CALL KEYPTnnnnvvON i

nnnn = Keypoint name
vv = version number
ON = an optional statement
i = processor ID (optional)

The Call Keypoint cards must be preceded by a Load Keypoint card. These cards specify the names and version numbers of keypoints in columns 21–26. The valid keypoints are:

CTK0 Keypoint 0
CTK1 Keypoint 1
CTK2 Keypoint 2
CTK3 Keypoint 3 (Reserved for customer use)
CTK4 Keypoint 4 (Reserved for customer use)
CTK5 Keypoint 5
CTK6 Keypoint 6
CTK9 Keypoint 9
CTKA Keypoint A
CTKB Keypoint B
CTKC Keypoint C
CTKD Keypoint D
CTKE Keypoint E
CTKI Keypoint I
CTKM Keypoint M
CTKV Keypoint V

The specified keypoints are written to the general file, in an area that is reserved for online keypoints. ACPL copies the keypoints from the general file to the working keypoint area if the keypoints have not been called by a Call General File Keypoint card.

Each card sets a switch in the loader control record to indicate that the keypoint was loaded. The general file loader online segment uses this information when writing the keypoints to the online modules.

In a loosely coupled environment, when a processor-unique keypoint is being loaded, the processor ID is specified in column 30. If column 30 is blank, the default

is the ID of the first processor in the system. When shared keypoints are being loaded, column 30 can be blank or any valid processor ID.

Leaving columns 27 and 28 blank on the Call Keypoint card (as in the example that follows) causes any associated REP card information to patch the Call Keypoint card information before the keypoint is loaded.

```
LOADER CALL KEYPTCTKA40 C <-- Loads keypoint CTKA
REP 000348 0014700 <-- Patches the new keypoint before loading --> CKTA40
```

Placing ON in columns 27 and 28 of the Call Keypoint card (as in the example that follows) causes the associated REP card information to be written to the *online* keypoint without loading the keypoint to the general file.

```
LOADER CALL KEYPTCTKA40ON C <-- No load of keypoint CTKA
REP 000348 0014700 <-- Patches applied to online keypoint CTKA --> CKTA40
```

Note: At least one REP card must follow the Call Keypoint card when using the ON facility.

Patching:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

```
zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 nnnn = Associated keypoint name
 vv = version number

Each REP card must contain the same name and version number in card columns 75–80 as that punched in card columns 21–26 of the corresponding Call Keypoint card.

PROG-MOD-BASE Clear Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

```
LOADER PROG-MOD-BASE CLEAR
```

The PROG-MOD-BASE Clear card resets the master extra program record so that all extra program records (#XPRG1 fixed file records) are available to be dispensed. If the Clear card is present, ALDR sets an indicator for ACPL to initialize the master extra program record. This occurs on image 1.

The card is optional and can be placed anywhere **after** the LOAD CP card and **before** the LOAD OPL and LOAD AP cards. The card must be included to initialize a program base before any C load modules are loaded to that image.

Notes:

1. All C load modules must be reloaded when the LOADER PROG-MOD-BASE CLEAR card is included.
2. If the LOADER PROG-MOD-BASE CLEAR card is added while a ZOLDR ACCEPT is in progress for a loadset that contains a C load module, then the ELDR Clear card must also be included in the load deck.

ELDR Clear Card:


```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER ELDR CLEAR
```

The ELDR Clear card ensures clean E-type loader record definitions. If it is present, ALDR will set an indicator for ACPL to clear and reinitialize all E-type loader records for image one. To clear E-type loader records in other images, use the auxiliary loader.

Attention: The ELDR Clear card will cause all E-type programs that were loaded or activated (but not accepted) with the E-type loader to be cleared on the online system.

Load OPL Card and PARS Card:

Load OPL Card

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER LOAD OPL LIST
```

LIST = an optional parameter that causes the general file loader to print the name and version number of each program loaded.

PARS Card

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
PARSvv
```

vv = PARS list version

The Load OPL card causes E-type programs to be loaded to the general file. The programs are defined by one or more operational program list (OPL) records. Each of these records contains a series of segment names and version numbers. Each of these records resides on the object module library with the name PARS and a unique 2-character version number. Different lists are indicated by different PARS cards.

If the Load OPL card is not followed by a PARS card then the program uses PARS00 as the list of OPL programs. If one or more PARS cards are supplied, the version numbers specified are used.

The optional operand LIST in card columns 31–34 of the Load OPL card causes the general file loader to print the name and version number of each program loaded.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnv
```

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

nnnn = Segment name

vv = version number

One or more REP cards can follow the PARS cards or, if no PARS cards are present, the Load OPL card. Each REP card must contain the segment name and version number in card columns 75–80. Although the offline segment of the general file loader will handle patch cards for different segments in any sequence and freely

intermixed, the execution time will be minimized if all REP cards for a segment are in one group. This eliminates needless I/O operations.

Note: REP cards for C load modules are not supported. If a REP card is specified for a C load module, a warning message is issued and the REP card is ignored.

Load AP Card and Call Program Card:

Load AP Card
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD AP LIST

LIST = an optional parameter that causes the general file loader to print the name and version of each program loaded.

Call Program Card
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER CALL PROG nnnnvv

nnnn = program name
vv = version number

The Load AP card tells the general file loader offline segment to accept Call Program cards for the selective loading of E-type programs. Any program loaded by this function will overlay any other version of the same program already loaded from an OPL record.

The optional operand LIST in card columns 31–34 of the Load AP card causes the general file loader to print the name and version number of each program loaded.

Note: If there is a Call Program card syntax error, that program will not load. Also, all cards that follow the incorrect card (with the same Load AP card) will not load.

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
nnnn = Segment name
vv = version number

One or more REP cards can follow a Call Program card. Each REP card must contain the same name and version number in card columns 75–80 as that punched in card columns 21–26 of the corresponding Call Program card.

Note: REP cards for C load modules are not supported. If a REP card is specified for a C load module, a warning message is issued and the REP card is ignored.

Load ACPL Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER   LOAD  ACPL vv
```

vv = version number

When the program encounters the Load ACPL card, the general file loader online segment is loaded to the general file as core-image records. Its file address on the general file is generated from its ordinal number obtained from CTKX. Switches are set in the loader control record to indicate to the general file loader online segment that the online segment was loaded.

Note: ACPL can only be loaded for BSS.

Patching:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx  ACPLvv
```

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

vv = version number

One or more REP cards can follow the Load ACPL card. Each REP card must contain ACPL in card columns 75–78 and the version number in columns 79–80.

Load ICDF Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER   LOAD  ICDF vv
```

vv = version number

The Load ICDF card causes the in-core dump formatter (ICDF) to be loaded to the general file as core-image records. Its file address on the general file is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that ICDF was loaded.

Note: ICDF can only be loaded for BSS.

Patching:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx  ICDFvv
```

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

vv = version number

One or more REP cards can follow the Load ICDF card. Each REP card must contain ICDF in card columns 75–78 and the version number in columns 79–80.

Load FCTB Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD FCTB vv

vv = version number

The Load FCTB card causes the FACE table to be loaded to the general file as core image records. Its file address on the general file is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that the FCTB was loaded. If the FCTB is to be loaded in program object format from the hierarchical file system (HFS) under OS/390 UNIX System Services (OS/390 UNIX), specify the name of a search path that was previously initialized in the load deck by a Path card. The format of the Load FCTB card, including the optional path name, follows.

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD FCTB vv/pathname

vv = version number

pathname = variable length alphanumeric name assigned by user to search path

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx FCTBvv

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

vv = version number

Only data in the first 16 MB of the FCTB can be patched.

Load SIGT Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD SIGT vv

vv = version number

The Load SIGT card causes the global synchronization table (SIGT) to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that SIGT was loaded.

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx SIGTvv

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

vv = version number

One or more REP cards can follow the Load SIGT card. Each REP card must contain SIGT in card columns 75–78 and the version number in columns 79–80.

Load USR1 Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER   LOAD   USR1 vv
```

vv = version number

The Load USR1 card causes the first user-defined CIMR area (USR1) to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. SIP automatically generates one ordinal number for USR1. A switch is set in the loader control record to indicate to the general file loader online segment that USR1 was loaded. The load USR1 card is required for a full load.

Note: USR1 can only be loaded for BSS.

Patching: One or more REP cards can follow the Load USR1 card. Each REP card must contain USR1 in card columns 75–78 and the version number in columns 79–80.

Load USR2 Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER   LOAD   USR2 vv
```

vv = version number

The Load USR2 card causes the second user defined CIMR area (USR2) to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. SIP automatically generates one ordinal number for USR2. A switch is set in the loader control record to indicate to the general file loader online segment that USR2 was loaded. The load USR2 card is required for a full load.

Patching: One or more REP cards can follow the Load USR2 card. Each REP card must contain USR2 in card columns 75–78 and the version number in columns 79–80.

Load IPLA Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER   LOAD   IPLA vv
```

vv = version number

The Load IPLA card causes program IPLA to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that IPLA was loaded.

Notes:

1. This is not the IPLA that is used on the general file IPL. The general file IPL uses the IPLA records that were written to the general file by the ICKDSF MVS program.
2. IPLA can only be loaded for BSS.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx IPLAvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load IPLA card. Each REP card must contain IPLA in card columns 75–78 and the version number in columns 79–80.

Load IPLB Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  LOAD  IPLB vv
```

vv = version number

The Load IPLB card causes program IPLB to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that IPLB was loaded.

Note: IPLB can only be loaded for BSS.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx IPLBvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load IPLB card. Each REP card must contain IPLB in card columns 75–78 and the version number in columns 79–80.

Load RIAT Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  LOAD  RIAT vv
```

vv = version number

The Load RIAT card causes the record identifier attribute table (RIAT) to be loaded to the general file as core-image records. Its file address is generated from its ordinal number obtained from CTKX. A switch is set in the loader control record to indicate to the general file loader online segment that RIAT was loaded.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx RIATvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load RIAT card. Each REP card must contain RIAT in card columns 75–78 and the version number in columns 79–80.

LDT Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LDT

This must always be the last control card for a given run and stops processing. If no LDT card is present, a message is printed, stating that the run cannot be completed.

Path Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LOADER	PATH	pathname	optional comments
		searchpath	

pathname = alphanumeric name, up to 16 characters in length, used to reference the search path
 searchpath = fully qualified search path occupying up to 14 input cards, as described below

The optional Path card actually consists of two or more cards in the input deck, and is used to initialize a search path that can be referenced on a subsequent Load FCTB card to load an FCTB, in program object format, from the hierarchical file system (HFS) under OS/390 UNIX System Services (OS/390 UNIX). The first card specifies the name that can be used on a subsequent Load FCTB card to reference the search path. After the actual Path card, as many as 14 additional input cards can be included to specify directories in the search path.

One or more absolute directories can be specified starting with the first directory to be searched and ending with the last directory to be searched. If more than one directory is specified, a ; symbol must separate directories. Blanks that follow a ; symbol are ignored. The presence of any character other than a blank in column 80 indicates that the search path continues starting in column 10 on the next card in the load deck. If there is no continuation character in column 80, the path will include all characters starting in column 10 and ending with the last nonblank on the input card. Comments can be included on the Path card starting in column 33. However, comments cannot be included on the subsequent cards that specify the actual search path because all nonblank characters are considered part of the path. An example of a Path card and continuation card that is used to specify a search path for the FCTB follows:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LOADER	PATH	FCTBPATH	Defining FCTBPATH for later user	
			/u/tpf41/produt_components/system/;	X
			/u/tpf41/product_components/test/;	X
			/u/user1/my_project/tables;	X
			/u/tpf41/my_project/phase_one/beta_test_versions/file_address_compute_X	
			table/program_object	

If the Path card shown in the previous example was specified before a Load FCTB card for FCTB40 that specifies FCTBPATH in the load deck, TPFLDR would then attempt to locate the FCTB as a program object file by searching for the following files, in order:

1. /u/tpf41/product_components/system/FCTB40
2. /u/tpf41/product_components/test/FCTB40
3. /u/user1/my_project/tables/FCTB40
4. /u/tpf41/my_project/phase_one/beta_test_versions/file_address_compute_table/program_object/FCTB40

Note: A path name can be defined only once in the load deck. The Path card defining a path name must precede the Load FCTB card that references the path name.

Loading System Components to the General File

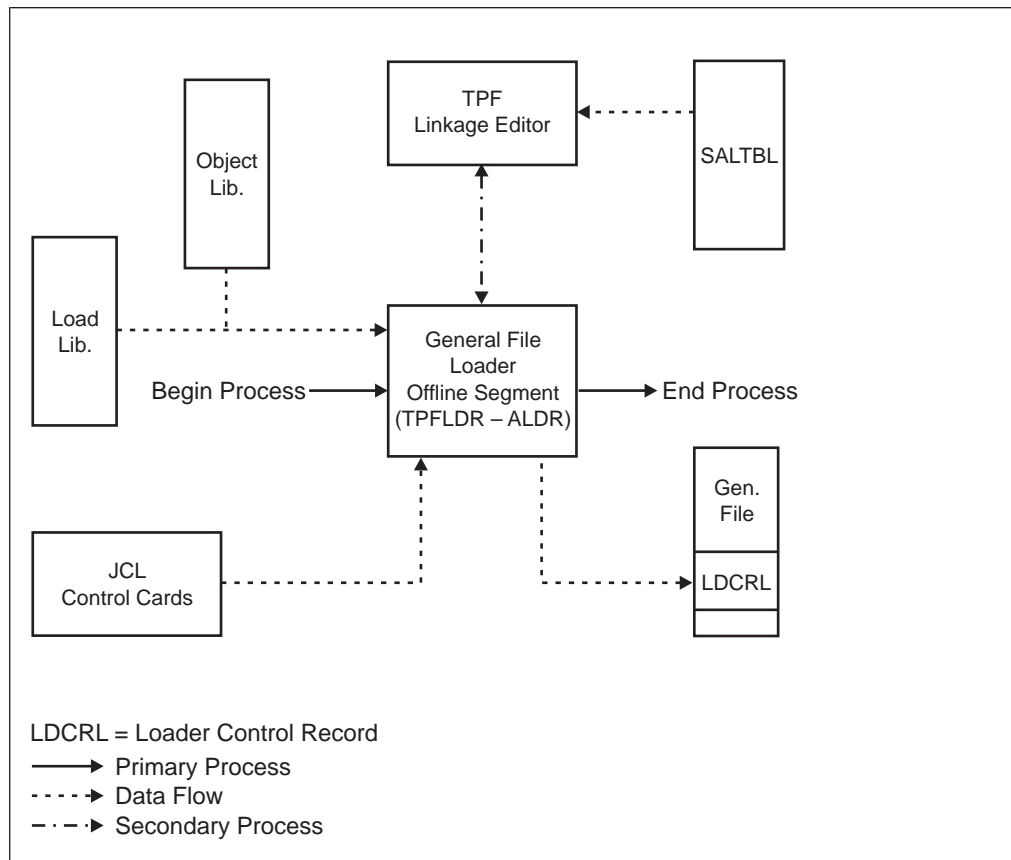


Figure 16. General File Load Using the General File Loader Offline Segment (ALDR)

When you create your TPF system for the first time, you must format online modules, initialize and format the general file, and load IPLA and the volume identifier (VOLID) to the general file. See "Initializing the General File" on page 322 and "Formatting the General File and Online Modules" on page 322 for more information.

The remainder of the general file is loaded by the general file loader offline segment (ALDR). ALDR activates the TPF linkage editor (LEDT/NLDT) to link the E-type programs with input from the system allocator table (SALTBL). ALDR then loads all the records and programs specified in the load deck to the general file.

ALDR also creates a loader control record (LDCRL) on the general file. This control record shows the general file loader online segment (ACPL) the components to load from the general file to the online modules.

Output Listing

An output listing is produced which details all system components loaded to the general file. Errors which do not cause the program to abend are also written in this listing. Abend error messages and informational messages are sent to the operator console. For more information on error messages see *Messages (System Error and Offline)* and *Messages (Online)*.

IPLing the General File

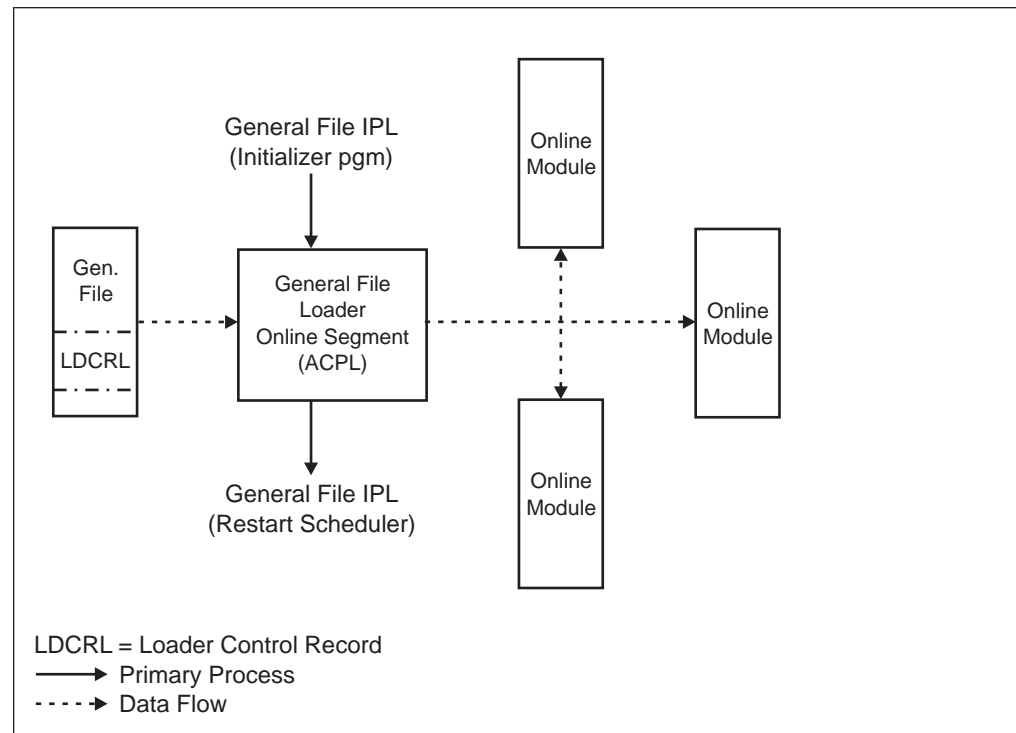


Figure 17. General File IPL Using the General File Loader Online Segment (ACPL)

When you IPL the general file, the general file loader online segment (ACPL) is placed in main storage. ACPL uses the loader control record (LDCRL) segment to put the contents of the general file onto online modules.

Keypoints are loaded to the prime and duplicate modules. They are copied to the first 256 modules of the same type as the prime module during online execution.

At the completion of the online loading, the operator is notified about all components loaded. If the IPL is unsuccessful, the operator is sent all of the error messages for the run.

Note: See *Messages (System Error and Offline)* and *Messages (Online)* for a complete listing of error and status messages.

If the IPL is successful, the general file loader sends the following message to the operator's terminal:

ACPL0001I SYSTEM LOAD COMPLETE
LOAD DATA AFTER 1052 STATE IF NEEDED
IF NOT IPL PRIME MODULE *ccud*

This message reminds an operator performing a full load to load data after the system announces it is in 1052 state (see “Loading Fixed-File Records”).

Loading Fixed-File Records

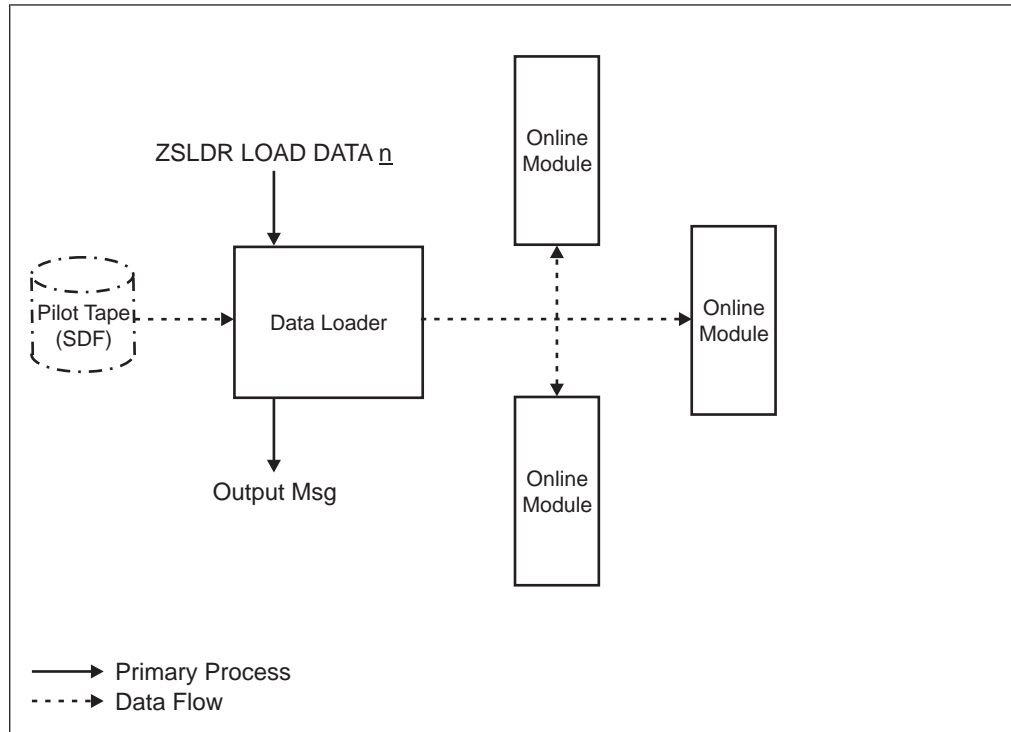


Figure 18. Loading Fixed-File Records Using the ZSLDR Command

The first time a system has been brought to 1052 state by a general file IPL it should be loaded, using the data loader, with fixed-file records. These records are used by the system and by the application programs.

To load fixed-file records, do the following:

1. Create a pilot tape (symbolic name = SDF) with the system test compiler (STC) program. See *TPF Program Development Support Reference* for information on the system test compiler program.

Note: A maximum of 65 535 records can be read or written from a pilot tape.

2. Mount the pilot tape (SDF). This is the only input to the data loader.
3. Enter the ZSLDR command to activate the data loader. See *TPF Operations* for more information about the ZSLDR command.

Note: Once records are loaded to the TPF database, no utility is available to remove the records from the database. If it is necessary to delete previously loaded records from the TPF database, you must create and load a new pilot tape that contains dummy records. If the number of

records decreases from one load to the next, the additional records from the first load will **not** be removed from the database.

4. Either of 2 actions can now be taken.
 - a. If the loaded records are meant for main storage, an online module IPL must be performed to place the records there.
 - b. The system can be cycled to NORM state without an online module IPL. An IPL will be required in the future to place the records in main storage.

Notes:

1. The data loader can be operated only while the TPF system is in 1052 state unless the ID of the pilot tape is N. If the ID of the pilot tape is N, the TPF system can be in any state.
2. In a loosely coupled facility, the subsystem cannot be above 1052 state in any processor unless the ID of the pilot tape is N. If the ID of the pilot tape is N, the subsystem can be in any state.

Loading System Components to an Existing TPF System

The auxiliary loader is the main method for performing system loads for an existing TPF system. The general file loader is only necessary at system generation time or if an emergency load condition in which no fallback image exists.

Use the auxiliary loader to load system components from a storage medium to any defined and disabled TPF image. Because the auxiliary loader operates in 1052 state or above, the subsystem must be initialized through a full load before the auxiliary loader can be used. The auxiliary loader requires only a hard IPL to switch the now active image.

Multiple Image Users

If you are running more than one TPF image, make sure the FCTBs and RIATs for each system have compatible structures. That is, make sure the tables:

- Do not point to different addresses for the same record.
- Do not point to the same address for different records.

Perform the following steps to load system components to an existing TPF system:

1. Create a new fallback image.
2. Create an auxiliary load deck.
3. Load system components to a storage medium.
4. Load the storage medium to the system.
5. Enable the image.
6. Move keypoint to the #KEYPT working area (optional, see “Moving Keypoints to the Working Area” on page 361).
7. IPL the image.

Creating a New Fallback Image

To load system components to a fallback image, you must first have a fallback image. If you do not have a fallback image, or want to create a new fallback image, you must do the following:

1. Define the fallback image.

2. Copy CTKX, CIMR components, IPL areas, and program areas from another image to the new fallback image.

Defining a Fallback Image

To define a new image (for example, TPF02) enter **ZIMAG DEF TPF02 NUM 2 IPL x PROG y**.

Note: Where x is the IPL area and y is the PROG area for TPF02.

Copying CTKX, IPL Areas, Program Areas, and CIMR Components

To physically copy CTKX, IPL areas, program areas, and CIMR components to a fallback image, do the following:

1. Enter **ZIMAG COPY TPF01 TPF02 CTKX** to physically copy CTKX from image TPF01 to image TPF02,

Note: CTKX must be loaded or copied to an image before any CIMR components can be copied.

2. Enter **ZIMAG COPY TPF01 TPF02 IPL** to physically copy IPL areas from image TPF01 to image TPF02,
3. Enter **ZIMAG COPY TPF01 TPF02 PROG** to physically copy program areas from image TPF01 to image TPF02,
4. Enter **ZIMAG COPY TPF01 TPF02 COMP ALL PHYSICAL** to physically copy all CIMR components from image TPF01 to image TPF02,

To logically copy CIMR components from image TPF01 to image TPF02, use the **ZIMAG COPY LOGICAL** command. For example, enter **ZIMAG COPY TPF01 TPF02 COMP FCTB.ICDF.ACPL.SIGT.RIAT.USR1.USR2 LOGICAL**, assuming that you are loading a new CPS0.

Notes:

1. The CTKX, IPL areas, and program areas cannot be logically copied.
2. A logical copy cannot be made of the IPAT if the two images are using different program areas.

Creating an Auxiliary Load Deck

This section shows the steps to take to create a load deck.

Enter the JCL Cards

The following JCL cards are used to run the auxiliary loader offline segment (TLDR); *nn* is the release identification and *ssid* is the subsystem ID.

```

//TLDRVTAP JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A
//TLDR EXEC PGM=TPFLDRnn,REGION=7000K,PARM='TLDR,SYS=ACP,CLMSIZE=4000000'
//STEPLIB DD DSN=ACP.LINK.RELnn ssid,DISP=SHR
// DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
// DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SALTB DD DSN=ACP.SALTB.LRELnn ssid,DISP=SHR
//OBJDD DD DSN=ACP.OBJ.RELnn ssid,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn ssid,DISP=SHR
//ALDRCL DD DSN=ACP.LINK.RELnn ssid,DISP=SHR
//ADATAIN DD PDS=xxx
//ADATATMP DD DSN=&&SYSUT1,SPACE=(TRK,(100,20)),UNIT=SYSDA
//TLDROUT DD UNIT=SYSDA,DSN=&&TLDROUT,DISP=(NEW,PASS)
//TAPEOUT DD DSN=&&SYSUT1,SPACE=(TRK,(100,20)),UNIT=SYSDA
//TEMPLOAD DD DSN=&&SYSUT1,SPACE=(TRK,(100,20)),UNIT=SYSDA
//LOADLIST DD SYSOUT=A
//LOADSUM DD DSN=&&LOADSUM,DISP=(NEW,PASS),UNIT=SYSDA,
// LRECL=133,SPACE=(TRK,(10,10)),RECFM=FBA
//SYSUDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//LDRTRACE DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
/*
//MKTAPE EXEC PGM=IEBGENER,REGION=58K
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD DSN=TLDR.TAPE,LABEL=(,SL),UNIT=3480,
// VOL=SER=vvvvvv,DISP=(,KEEP),
// DCB=(BLKSIZE=4095,RECFM=U)
//SYSUT1 DD DSN=&&TLDROUT,UNIT=SYSDA,DISP=(OLD,DELETE)
//PRTSUM EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=&&LOADSUM,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSUT2 DD SYSOUT=*,LRECL=133
//SYSIN DD *
PRINT PREFORM=A
/*

```

Figure 19. TLDR Run Load Deck (to Tape) Example

Notes:

1. DD names SALTB, OBJDD, TAPEOUT, TEMPLOAD, LOADLIST, TLDROUT, LOADSUM, SYSIN, and LOADMOD are required.
2. DD name LDRTRACE is required only if you specify TRACE(ON) as a parameter.
3. Data set concatenation is sequence dependent.
4. Figure 19 shows a TLDR load deck using tape as the storage medium. “JCL Load Deck Examples” on page 417 shows JCL load decks using other storage medium.
5. If the storage medium is tape, use standard label tapes.

TPFLDR Exec Card: The TPFLDR Exec card identifies the specified version of the TPF offline loader program (TPFLDR).

The following parameters can be specified on the EXEC card:

1. TLDR
indicates that the auxiliary loader (TLDR) is to be run.
2. PGMNBR = x

where *x* is the maximum number of program calls that can be done in the load. This number is used to GETMAIN an area for a program table. The current maximum value is 599 186. Coding a value greater than this will result in a PARM ERROR message. If not specified, the PGMNBR default is the value of the SALCOUNT field in the header of the offline SALTBL referred to by the SALVERS control card.

3. PTCHNBR = *x*

where *x* is the maximum number of REP cards that can be included in the load. This number is used to determine the storage area size for a patch table. The size of the area, in bytes, is 84 times the PTCHNBR. The current maximum value is 199 728. Coding a value greater than this will result in a PARM ERROR message. If not specified, the PTCHNBR default is 800 REP cards.

4. BUFSIZE = *x*

where *x* is the maximum size of any component that is included in the load deck. The default value is 2 000 000. Check the size of the IPAT, FCTB, and CP to determine the correct value for this parameter. Any number up to 7 digits is allowed.

Note: When using the PGMNBR, PTCHNBR, and BUFSIZE defaults, specify a REGION of at least 7000K on the EXEC card.

5. SYS = *nnn*

where *nnn* is a 3-character ID printed in the load listing and load summary. The default is blanks.

6. CLMSIZE = *x*

where *x* is greater than or equal to the MVS file size of the largest C load module to be loaded. If a C load module contains a higher than normal ratio of VCONs to executable code, CLMSIZE must be larger. TLDR searches only the OBJDD DSN (not LOADMOD DSN) when CLMSIZE = 0.

7. ADATASIZE = *x*

where *x* is greater than or equal to the MVS file size of the largest ADATA file to be loaded. ADATA does not attempt to load ADATA files when ADATASIZE=0.

8. TRACE(ON|OFF)

If you specify TRACE(ON), selective trace data will be written to the LDRTRACE DD name. This trace data can be useful to system programmers to debug problems that occur while running TPFLDR. TRACE(ON) is primarily intended for TPF development personnel for this purpose. TRACE(OFF) is the default. If you specify TRACE(OFF), no trace data is written.

STEPLIB Card: The STEPLIB card identifies the partitioned data sets that contain the offline loader programs.

SALTB Card: The SALTB card identifies partitioned data sets that contain the system allocator table (SALTBL). The SALTBL contains information necessary for the TPF linkage editor to resolve virtual address constants (VCONs). These VCONs can be program addresses, tape names, macro parameters, and others that are put in the SALTBL to allow relocation of resources in the online system. See "System Allocator" on page 399 for more information on the SALTBL.

OBJDD Card: The OBJDD card identifies object libraries. Object libraries are partitioned data sets of assembled programs. All programs to be loaded must be in object libraries with the exception of the control program, FACE table, and C load modules. Library contents are maintained and updated using MVS procedures. For more information, see *TPF System Generation*.

LOADMOD Card: The LOADMOD card identifies the partitioned data set (LOADMOD) that contains the link-edited C load modules. When an E-type program is being loaded, the data sets identified by the LOADMOD card are searched before the data sets identified by the OBJDD card are searched. The C load modules are compiled and link-edited using standard MVS facilities. For more information, see *TPF System Generation*.

ALDRCPPL Card: The ALDRCPPL card identifies the partitioned data set that contains the link-edited control program. The control program is assembled and link-edited using standard MVS facilities. For more information see *TPF System Generation*.

ADATAIN Card: The optional ADATAIN card identifies the partitioned data set (PDS) that contains ADATA files to be loaded with real-time programs. If the ADATAIN card is present, TLDR will attempt to find ADATA files in the data set specified by the ADATAIN card. SYSADATA files are generated by the high level assembler (HLASM) and must be processed by the SYSADATA postprocessor (program TPFDBG) to create ADATA files before they can be loaded. When TLDR loads a basic assembly language (BAL) program, it will attempt to find an ADATA file by searching for a PDS member with a matching name. When TLDR loads an E-type load module, it will attempt to find an ADATA file for each object module linked into the load module. TLDR will ensure that the assembly date of the ADATA file matches the assembly date of the assembler language program before loading an ADATA file.

Note: If a program was assembled more than once on a given date, it is possible for TLDR to load an incorrect ADATA file with an assembler language program. You must ensure this does not happen. Whenever an E-type program (BAL or load module) is loaded to the online system by the auxiliary loader, any ADATA file associated with a previously loaded version of that program is effectively erased. For more information, see *TPF System Generation*.

ADATATMP Card: The ADATATMP card specifies a temporary data set used to hold ADATA files before they are written to the output medium.

TLDROUT Card: The TLDROUT card identifies the output device to which offline programs and data are written.

TAPEOUT Card: If E-type programs are loaded, the TAPEOUT card creates program version records (PVRs) and writes them to the TAPEOUT file.

TEMPLOAD Card: The TEMPLOAD card reads load cards and checks the format. It adds program names to a name table and copies the input card image to temporary data sets used to generate the detail report.

LOADLIST Card: The LOADLIST card causes an output listing that details all system components loaded to the storage medium to be produced. Errors that do not cause the program to abend are also written in this listing. Abend error messages and informational messages are sent to the operator console. For more information on error messages see *Messages (System Error and Offline)* and *Messages (Online)*.

LOADSUM Card: The LOADSUM card causes an output listing summary of the system components loaded to the storage medium.

Enter the Input Control Cards

Figure 20 shows a sample TLDR input control load deck. The example explains the order of the cards and states if the cards are required or optional. This order is not affected by deleting optional cards or adding comment cards.

```
=====
      Initialization part of the load deck - Must be first input control cards
=====
SYSID=BSS                                <-- Required, must be first
SALVERS=40                              <-- Required, immediately after SYSID
LOADER  LOAD  CTKX 40                    <-- Optional
=====
      Main part of the load deck - Must be second input control cards
=====
LOADER  LOAD  CP 40                      <-- Optional, before file-resident programs
LOADER  LOAD  IPAT 40                    <-- Optional, anywhere after LOAD CP
LOADER  LOAD  KEYPT                      <-- Optional, anywhere after LOAD CP
LOADER  CALL  KEYPTCTK040                 <-- Optional, immediately after LOAD KEYPT
LOADER  CALL  KEYPTCTK140
LOADER  CALL  KEYPTCTK241
LOADER  CALL  KEYPTCTK340
LOADER  CALL  KEYPTCTK440
LOADER  CALL  KEYPTCTK540
LOADER  CALL  KEYPTCTK640
LOADER  CALL  KEYPTCTK940
LOADER  CALL  KEYPTCTKA40
LOADER  CALL  KEYPTCTKB40
LOADER  CALL  KEYPTCTKC40
LOADER  CALL  KEYPTCTKD40
LOADER  CALL  KEYPTCTKE40
LOADER  CALL  KEYPTCTKI40
LOADER  CALL  KEYPTCTKM40
LOADER  CALL  KEYPTCTKV40
LOADER  CALL  KEYPTCTKA40 C
LOADER  PROG-MOD-BASE CLEAR              <-- Optional, anywhere after LOAD CP and LOAD OPL
LOADER  APRG CLEAR                      <-- Optional, anywhere after LOAD CP
LOADER  ELDR CLEAR                      <-- Optional, anywhere after LOAD CP
LOADER  LOAD  OPL                      <-- Optional, anywhere after ELDR CLEAR
PARS40                                  <-- Optional, immediately after LOAD OPL
PARS4D
PARSP1
LOADER  LOAD  AP                        <-- Optional, anywhere after LOAD CP
LOADER  CALL  PROG COSY40               <-- Optional, immediately after LOAD AP
LOADER  CALL  PROG BXAX40
LOADER  CALL  PROG CCKP40
LOADER  CALL  PROG COHA40
LOADER  CALL  PROG COHB40
LOADER  CALL  PROG UF00P1
REP 000348 0014700                    <-- Optional, immediately after CALL PROG -> UF00P1
LOADER  LOAD  ACPL 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  FCTB 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  ICDF 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  IPLA 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  IPLB 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  SIGT 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  RIAT 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  USR1 40                   <-- Optional, anywhere after LOAD CP
LOADER  LOAD  USR2 40                   <-- Optional, anywhere after LOAD CP
LDT                                     <-- Required, must be last
/*
```

Figure 20. TLDR Input Control Load Deck Example

Figure 21 shows a sample TLDR input control load deck that makes use of the PATH card to load an FCTB in program object format from an HFS path.

```
=====
      Initialization part of the load deck - Must be first input control cards
=====
SYSID=BSS                      <-- Required, must be first
SALVERS=40                     <-- Required, immediately after SYSID
LOADER  LOAD  CTKX 40          <-- Optional
=====
      Main part of the load deck - Must be second input control cards
=====
LOADER  PATH  FCTBPATH          This search path can be used to find FCTB
      /u/tpf41/product_components/system/;                      X
      /u/tpf41/product_components/test/;                        X
      /u/user1/my_project/tables
LOADER  LOAD  FCTB 40/FCTBPATH <-- FCTBPATH indicates search path to use
      LDT                      <-- Required, must be last
/*
```

Figure 21. TLDR Input Control Load Deck Example Showing FCTB in Program Object Format from HFS

All following cards in the same section (for example, LOADER LOAD KEYPT section or LOADER LOAD AP section) will be marked as *unknown card* and ignored when a card format error or sequence error is detected.

The IMAGE CLEAR card is not supported. A warning message will be generated if it is included in the load deck. Use the ZIMAG CLEAR command to initialize an image.

Comment Card:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
*comments
```

Comment cards can be placed anywhere in the load deck. Comments are identified by an asterisk (*) in card column 1.

PROG-MOD-BASE Clear Card:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
LOADER  PROG-MOD-BASE CLEAR
```

The PROG-MOD-BASE clear card resets the master extra program record so that all extra program records (#XPRGn fixed file records) are available to be dispensed. If the clear card is present, TLDR sets an indicator for the online part of the auxiliary loader to initialize the master extra program record. This occurs on the target image.

The card is optional and can be placed anywhere **after** the LOAD CP card and **before** the LOAD OPL and LOAD AP cards. The card must be included to initialize a program base before any C load modules are loaded to that image.

Notes:

1. All C load modules must be reloaded when the PROG-MOD-BASE clear card is included.

2. If the LOADER PROG-MOD-BASE CLEAR card is added while a ZOLDR ACCEPT is in progress for a loadset that contains a C load module, then the ELDR Clear card must also be included in the load deck.

APRGn Record Clear Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LOADER APRG CLEAR

The APRGn Record Clear card resets the master APRGn record so that all APRGn records (#APRGn fixed file records) are available to be dispensed. If the Clear card is present, TLDR sets an indicator for the online part of the auxiliary loader to initialize the master APRGn record. This occurs on the target image.

The card is optional and can be placed anywhere **after** the LOAD CP card and **before** the LOAD OPL and LOAD AP cards. The card must be included to initialize newly allocated APRGn records before ADATA files can be loaded with any assembler language programs.

Notes:

1. All online ADATA files are effectively erased when the LOADER APRG CLEAR card is included; if ADATA files are required to be online for any programs, those programs must be reloaded.

ELDR Clear Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

LOADER ELDR CLEAR

The ELDR Clear card ensures clean E-type loader record definitions. If it is present, TLDR will set an indicator for the online part of the auxiliary loader to clear and reinitialize all E-type loader records for the target image.

Attention: The ELDR Clear card will cause all E-type programs that were loaded or activated (but not accepted) with the E-type loader to be cleared on the online system.

Subsystem ID Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

SYSID=nnnn

nnnn = Subsystem ID

The Subsystem ID card must be the first noncomment card in the deck. It identifies the subsystem to be loaded. This ID is also the prefix of the command that activates the online segment.

System Allocator Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

SALVERS=vv

vv = version

The System Allocator card specifies the version of the SALTBL to be used for link-editing. The program will terminate if this is not the second non-comment card in the deck.

Load CTKX Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER  LOAD  CTKX vv
```

vv = version number

The Load CTKX card causes the image pointer record (CTKX) to be loaded to a storage medium in a 4KB record.

Patching: One or more REP cards can follow the Load CTKX card.

Load Control Program Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER  LOAD  CP   vv
```

vv = version number

The Load Control Program card is valid only for a load to the basic subsystem (BSS). The offline segment builds a copy of the control program in main storage from the load module library. The CP is then filed on a storage medium as a chain of 4KB records. A version number must be specified in columns 21–22.

Patching:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx  ccccccvv
```

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

cccccc = CP CSECT name

vv = version number

Patch the control program by placing REP cards after the Load Control Program card. Control program patching is done by patching CP CSECT individually, starting at relative address zero. Each REP card must contain the CP CSECT name starting in column 73.

Load IPAT Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
```

```
LOADER  LOAD  IPAT vv
```

vv = version number

The Load IPAT card causes the program allocation table (PAT) to be loaded to a storage medium as core-image records. The time stamp in the header of system allocator table (SALTBL) will be used to validate the IPAT. If the time stamp in the header of the IPAT does not match the time stamp of the system allocator table (SALTBL), the load will end.

Patching: One or more REP cards can follow the Load IPAT card. Each REP card must contain the IPAT in card columns 75–78 and the version number in columns 79–80.

Load Keypoint Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  LOAD  KEYPT
```

The Load Keypoint card tells the offline segment that there are keypoints to be loaded. The keypoints are identified by Call Keypoint cards or Patch Keypoint cards which must follow the Load Keypoint card.

Note: Be careful when you enter ZIMAG KEYPT MOVE to move loaded keypoints into the working area (#KEYPT). ZIMAG KEYPT MOVE affects all images and can, for certain keypoints, affect all loosely coupled processors. If a mistake is made, the original contents of the keypoints can be recovered with the ZIMAG KEYPT RESTORE command. Online updates to keypoint data, made between the MOVE and the RESTORE, will be lost.

Call Keypoint Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  CALL  KEYPTnnnnvvON i
```

```
nnnn = Keypoint name
vv    = version number
ON    = an optional statement
i     = processor ID (optional)
```

The Call Keypoint card specifies the name and version number of a keypoint in columns 21–26.

Note: See “Call Online Keypoint Card” on page 329 for a list of keypoint names.

The keypoint specified is written to the a storage medium. This new keypoint will be written to the keypoint staging area of the target image. After the image is enabled, the keypoints can be moved to their working areas with the ZIMAG KEYPT MOVE command.

In a loosely coupled environment, when a processor-unique keypoint is being loaded, the processor ID is specified in column 30. If column 30 is blank, the default is the ID of the first processor in the system. When shared keypoints are being loaded, column 30 can be blank or any valid processor ID.

Leaving columns 27 and 28 blank on the Call Keypoint card (as in the example that follows) causes any associated REP card information to patch the Call Keypoint card information before the keypoint is loaded.

```
LOADER  CALL  KEYPTCTKA40 C    <-- Loads keypoint CTKA
REP 000348 0014700    <-- Patches the new keypoint before loading -->  CKTA40
```

Placing ON in columns 27 and 28 of the Call Keypoint card (as in the example that follows) causes the associated REP card information to be written to the *online* keypoint without loading the new keypoint.

```
LOADER  CALL  KEYPTCTKA40ON C    <-- No load of keypoint CTKA
REP 000348 0014700    <-- Patches applied to online keypoint CTKA -->  CKTA40
```

Note: At least one REP card must follow the Call Keypoint card when using the ON facility.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx  nnnnvv
```

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

nnnn = Associated keypoint name

vv = version number

Each REP card must contain the same name and version number in card columns 75–80 as that punched in card columns 21–26 of the corresponding Call Keypoint card.

Patch Keypoint Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER PATCH KEYPTnnnnvv i
```

nnnn = Keypoint name

vv = version number

ON = an optional statement

i = processor ID (optional)

The Patch Keypoint card specifies the name and version number of a keypoint in columns 21–26. The Patch Keypoint card performs the same function as the Call Keypoint card using the ON facility. The Patch Keypoint card can be used by the TLDR only. The Patch Keypoint card causes the associated REP card information to be written to the *online* keypoint without loading the new keypoint.

Note: As shown in the example that follows, at least one REP card must follow the Patch Keypoint card.

```
LOADER PATCH KEYPTCTKA40 C <-- No load of keypoint CTKA
REP 000348 0014700 <-- Patches applied to online keypoint CTKA --> CKTA40
```

The format of the REP card is identical to REP cards for Call Keypoint cards.

In a loosely coupled environment, when a processor-unique keypoint is being loaded, the processor ID is specified in column 30. If column 30 is blank, the default is the ID of the first processor in the system. When shared keypoints are being loaded, column 30 can be blank or any valid processor ID.

Note: Patch Keypoint cards cannot be used with Call Keypoint cards if both specify the same keypoint.

Load OPL Card and PARS Card:

Load OPL Card

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER LOAD OPL LIST
```

LIST = an optional parameter that causes the auxiliary file loader to print the name and version number of each program loaded.

PARS Card

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

PARSvv

vv = PARS list version

These cards cause E-type programs to be loaded to a storage medium. The programs are defined by one or more operational program list (OPL) records. Each OPL record contains a series of segment names and version numbers. Each of these records resides on the object module library with the name PARS and a unique 2-character version number. Different lists are indicated by different PARS cards. The Load OPL card must be followed by at least one PARS card.

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnvv

z = Any character except *

aaaaaa = Address of the patch area

xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.

nnnn = Segment name

vv = version number

One or more REP cards can follow the PARS cards. Each REP card must contain the segment name and version number in card columns 75–80. Although the program will handle patch cards for different segments in any sequence and freely intermixed, the execution time will be minimized if all REP cards for any given segment are in one group. This eliminates needless I/O operations.

Note: REP cards for C load modules are not supported. If a REP card is specified for a C load module, a warning message is issued and the REP card is ignored.

Load AP Card and Call Program Card:

Load AP Card

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD AP LIST

LIST = an optional parameter that causes the auxiliary loader to print the name and version of each program loaded.

Call Program Card

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER CALL PROG nnnvv

nnnn = program name

vv = version number

The Load AP card tells the auxiliary loader offline segment to accept Call Program cards for the selective loading of E-type programs. If there are different version codes for a program in the OPL and AP lists, the auxiliary loader loads the last version specified in the AP list.

Note: If there is a Call Program card syntax error, that program will not load. Also, all cards that follow the incorrect card (with the same Load AP card) will not load.

Patching:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx nnnnvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a
 time, separated by commas or blanks.
nnnn = Segment name
vv = version number

One or more REP cards can follow a Call Program card. Each REP card must contain the same name and version number in card columns 75–80 as that punched in card columns 21–26 of the corresponding Call Program card.

Note: REP cards for C load modules are not supported. If a REP card is specified for a C load module, a warning message is issued and the REP card is ignored.

Note: If you include a REP card to patch a real time program that will have an ADATA file loaded, the TPF Assembler Debugger for VisualAge Client may not display an accurate listing view of the program.

Load ACPL Card:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8

LOADER LOAD ACPL vv

vv = version number

When the auxiliary loader encounters the Load ACPL card, the general file loader online segment is loaded to a storage medium as core-image records.

Note: This program can only be loaded to a basic subsystem (BSS).

Patching:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx ACPLvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a
 time, separated by commas or blanks.
vv = version number

One or more REP cards can follow the Load ACPL card. Each REP card must contain ACPL in card columns 75–78 and the version number in columns 79–80.

Load ICDF Card:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8

LOADER LOAD ICDF vv

vv = version number

The Load ICDF card causes the in-core dump formatter (ICDF) to be loaded to a storage medium as core-image records.

Note: This program can only be loaded to a basic subsystem (BSS).

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx ICDFvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a
time, separated by commas or blanks.
vv = version number

One or more REP cards can follow the Load ICDF card. Each REP card must contain ICDF in card columns 75–78 and the version number in columns 79–80.

Load FCTB Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD FCTB vv

vv = version number

The Load FCTB card causes the FACE table to be loaded to a storage medium as core image records. If the FCTB is to be loaded in program object format from the hierarchical file system (HFS) under OS/390 UNIX System Services (OS/390 UNIX), specify the name of a search path that was previously initialized in the load deck by a Path card. The format of the Load FCTB card, including the optional path name, follows.

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD FCTB vv/pathname

vv = version number
pathname = variable length alphanumeric name assigned by user to search path

Patching:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx FCTBvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a
time, separated by commas or blanks.
vv = version number

Only data in the first 16 MB of the FCTB can be patched. One or more REP cards can follow the Load FCTB card. Each REP card must contain FCTB in card columns 75–78 and the version number in columns 79–80.

Load SIGT Card:

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

LOADER LOAD SIGT vv

vv = version number

The Load SIGT card causes the Global Synchronization Table (SIGT) to be loaded to a storage medium as core-image records.

Patching:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
zREP aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx SIGTvv

z = Any character except *
aaaaaa = Address of the patch area
xxxx = Patch data. The data is entered 2 hex bytes at a
 time, separated by commas or blanks.
vv = version number

One or more REP cards can follow the Load SIGT card. Each REP card must contain SIGT in card columns 75–78 and the version number in columns 79–80.

Load USR1 Card:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER LOAD USR1 vv

vv = version number

The Load USR1 card causes the first user-defined CIMR area (USR1) to be loaded to a storage medium as core-image records.

Note: This program can only be loaded to a basic subsystem (BSS).

Patching: One or more REP cards can follow the Load USR1 card. Each REP card must contain USR1 in card columns 75–78 and the version number in columns 79–80.

Load USR2 Card:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER LOAD USR2 vv

vv = version number

The Load USR2 card causes the second user defined CIMR area (USR2) to be loaded to a storage medium as core-image records.

Patching: One or more REP cards can follow the Load USR2 card. Each REP card must contain USR2 in card columns 75–78 and the version number in columns 79–80.

Load IPLA Card:

|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER LOAD IPLA vv

vv = version number

The Load IPLA card causes the IPL program, IPLA, to be loaded to a storage medium in 4KB records.

Note: This program can only be loaded to a basic subsystem (BSS).

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx IPLAvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load IPLA card. Each REP card must contain IPLA in card columns 75–78 and the version number in columns 79–80.

Load IPLB Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  LOAD  IPLB vv
```

vv = version number

The Load IPLB card causes the IPL program, IPLB, to be loaded to a storage medium in 4KB records.

Note: This program can only be loaded to a basic subsystem (BSS).

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx IPLBvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load IPLB card. Each REP card must contain IPLB in card columns 75–78 and the version number in columns 79–80.

Load RIAT Card:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
LOADER  LOAD  RIAT vv
```

vv = version number

The Load RIAT card causes the record identifier attribute table (RIAT) to be loaded to a storage medium as core-image records.

Patching:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
```

```
zREP  aaaaaa 001xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx RIATvv
```

z = Any character except *
 aaaaaa = Address of the patch area
 xxxx = Patch data. The data is entered 2 hex bytes at a time, separated by commas or blanks.
 vv = version number

One or more REP cards can follow the Load RIAT card. Each REP card must contain RIAT in card columns 75–78 and the version number in columns 79–80.

LDT Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LDT
```

This must always be the last control card for a given run and ends processing.

Path Card:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER  PATH  pathname          optional comments
          searchpath
```

pathname = alphanumeric name, up to 16 characters in length, used to reference the search path
searchpath = fully qualified search path occupying up to 14 input cards, as described below

The optional Path card actually consists of two or more cards in the input deck and is used to initialize a search path that can be referenced on a subsequent Load FCTB card to load an FCTB, in program object format, from the hierarchical file system (HFS) under OS/390 UNIX System Services (OS/390 UNIX). The first card specifies the name that can be used on a subsequent Load FCTB card to reference the search path. After the actual Path card, as many as 14 additional input cards can be included to specify directories in the search path.

One or more absolute directories can be specified starting with the first directory to be searched. If more than one directory is specified, a ; symbol must separate directories. The presence of any character other than a blank in column 80 indicates that the search path continues starting in column 10 on the next card in the load deck. If there is no continuation character in column 80, the path will include all characters starting in column 10 and ending with the last nonblank on the input card. Comments can be included on the Path card starting in column 33. However, comments cannot be included on the subsequent cards that specify the actual search path because all nonblank characters are considered part of the path. An example of a Path card and continuation card that is used to specify a search path for the FCTB follows:

```
|...+...1....+...2....+...3....+...4....+...5....+...6....+...7....+...8
LOADER  PATH  FCTBPATH          Defining FCTBPATH for later user
          /u/tpf41/produt_components/system/;                      X
          /u/tpf41/product_components/test/;                      X
          /u/user1/my_project/tables;                              X
          /u/tpf41/my_project/phase_one/beta_test_versions/file_address_compute_ X
          table/program_object
```

If the Path card shown in the previous example was specified before a Load FCTB card for FCTB40 that specifies FCTBPATH in the load deck, TPFLDR would then attempt to locate the FCTB as a program object file by searching for the following files in order:

1. /u/tpf41/product_components/system/FCTB40
2. /u/tpf41/product_components/test/FCTB40
3. /u/user1/my_project/tables/FCTB40
4. /u/tpf41/my_project/phase_one/beta_test_versions/file_address_compute_table/program_object/FCTB40

Note: A path name can be defined only once in the load deck. The Path card defining a path name must precede the Load FCTB card that references the path name.

Loading System Components to a Storage Medium

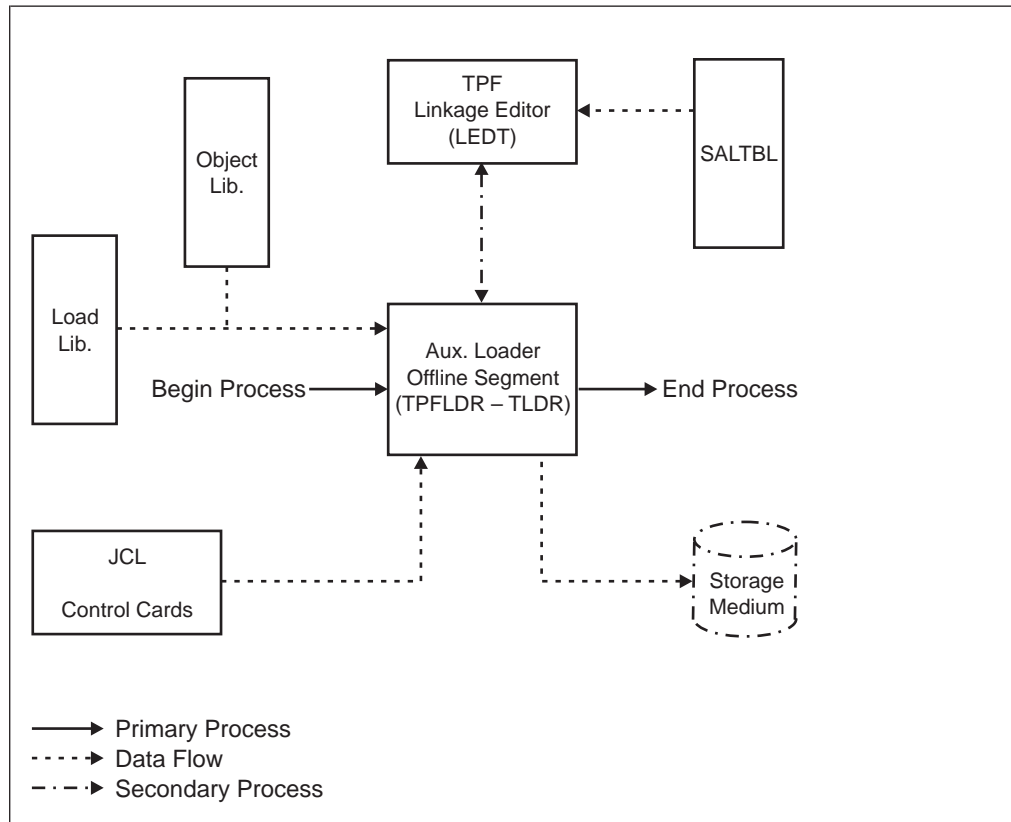


Figure 22. Auxiliary Load via Auxiliary Loader (Offline — MVS)

The auxiliary loader offline segment runs under MVS control and places the components to be loaded on a storage medium. The storage medium is then loaded to the system by the ZTPLD command. For more information about the ZTPLD command, see *TPF Operations*.

Note: If you are writing to a virtual reader, use the ELDRVRDR EXEC to convert TLDR JOB output to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

Output Listing

A load listing that details all programs and patches loaded, with any error messages, is produced by the offline segment. A load summary is also printed. In the load summary example that follows, items that are highlighted are explained in the **Key to Output Listing**. See *Messages (System Error and Offline)* and *Messages (Online)* for more information.

```

* LOAD SUMMARY *    DATE: 88.059    TIME: 15.40.48

  4   ACPDAB-OPL    2-ADESAB    AFGVAB-AP    BFDAAB-OPL
  8   CTK1AB-KPT    CVEHAB-OPL    DALBAB-OPL    DFREAB-AP
 12   2-IPLBKK      JCD0AB-OPL    KXOPAB-OPL    MIFFAB-OPL
 16   00ZECD-AP     PTV2CD-AP     QIE1CD-AP     WQXRCD-AP

```

ALLOCATOR VERSION = AB

THE FOLLOWING PROGRAMS WERE NOT LOADED

NAME	TYPE	REASON
CTKMAB	KPT	- KEYPT CONFIGURATION DEPENDENT
CTLBAB	OPL	- THIS AP PGM RECORD NOT ALLOWED
PREPCD	AP	- PGM NOT FOUND ON OBJ LIBRARY

THE FOLLOWING PROGRAMS HAVE WARNINGS

NAME	TYPE	WARNING
WQXRCD	AP	- PGM HAS UNRESOLVED V-CONS

LIST OF UNRESOLVED V-CONS

WQXRCD - WQX1 WQX2 WQX3 WQX4

Key to Output Listing

- 4** Number of programs loaded.
- ACPD** Program name.
- AB** Program version.
- OPL** Program was part of operational program list.
- 2** Number of patch cards applied (** in this location means 255 or more patches were applied).
- AP** Individually called program.
- KPT** Keypoint.

Figure 23. Sample Summary Listing for Auxiliary Loader (with Key to Output Listing)

Output Listing Return Codes

When inspecting the load listing returned after running the JCL job, there will be a TLDR return code. The return codes are shown below with an explanation and user response.

0

Explanation: No errors or warnings.

User Response: Load the components to a TPF image.

4

Explanation: Unknown card found in the AP/OPL section of the load deck.

User Response: Check for messages to the right of

the particular card in error. Some common errors are misspelling a card name, putting a card in the wrong sequence, or including a non-TLDR card. A return code of 4 is usually a warning, so you can still load the components to a TPF image.

8

Explanation: The E-type program was not loaded, patches for the E-type program were not applied, or a program call was superseded by a later call.

User Response: Check for messages to the right of the particular program being flagged.

Notes:

1. If an item is not found, make sure the required library is included in the TLDR JCL.
2. If the program has been superseded by a later call, check to make sure that the later version is the version you want and that it loaded successfully. If it is successful, you can load the components to a TPF image.

12

Explanation: Error processing the CP, keypoints, core image programs, or their patches.

User Response: Check for messages to the right of the item in error. The TLDR JCL may have created a correct module to load, but it is not complete as intended.

1000

Explanation: Error on a C function when attempting to open a file, read from a file, or write to a file.

User Response: Check for a message specifying the exact error that occurred. The SYSERR card must be included in the JCL to see the C function error messages. Determine and correct the problem with the file.

Loading from a Storage Medium to the TPF System

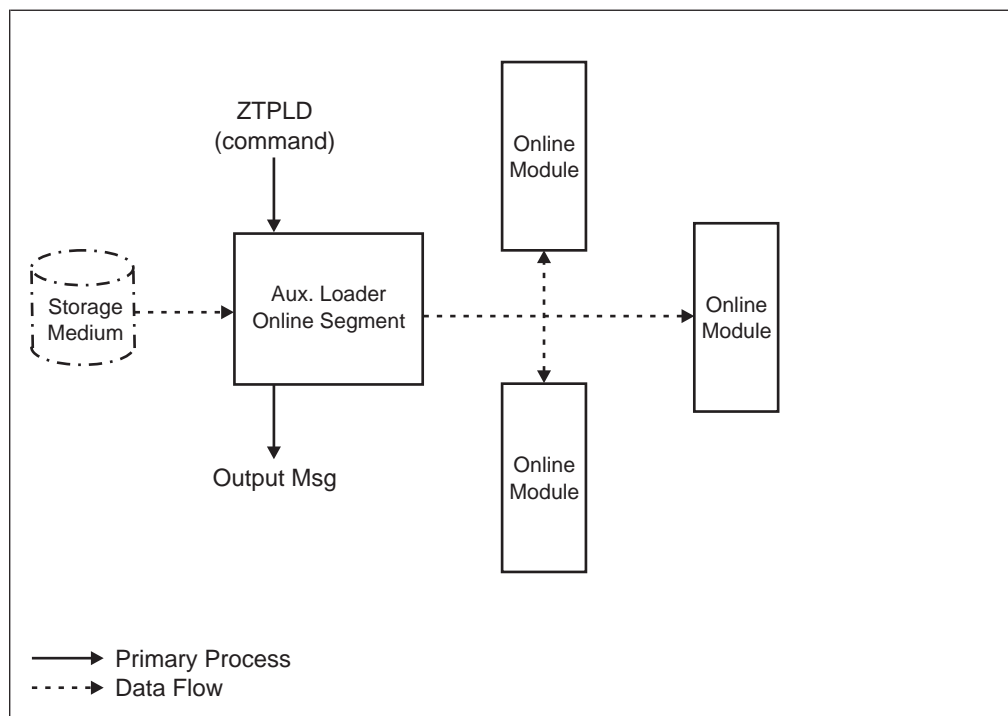


Figure 24. Auxiliary Load via Auxiliary Loader (Online — TPF)

The online segment is started by the ZTPLD command. For more information about this command, see *TPF Operations*.

Notes:

1. Use the ZTPLD command to load system components to a target image. The target image must be disabled and its IPL and PROG areas must not be referred to by any enabled image (if loading IPL or PROG components).

Use the ZIMAG DISABLE command to disable an image. Use the ZIMAG DISPLAY command to display the status of the IPL and PROG areas. See *TPF Operations* for more information about the use of these commands.

2. If you are loading from a virtual reader, make sure your TLDR job output has been converted. Use the ELDRVRDR EXEC to convert TLDR JOB output to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

The online segment sends the following message after a successful load:

```
TPLD0004I hh.mm.ss LOAD COMPLETE
```

where *hh.mm.ss* is a time stamp.

Additional status and error messages are also sent to the operator console. For more information about auxiliary loader messages, see *Messages (System Error and Offline)* and *Messages (Online)*.

Enabling an Image

Once an image is loaded, enter the ZIMAG ENABLE command to enable it. For example, to enable image TPF02, enter **ZIMAG ENABLE TPF02**. You will receive a message that tells you if the components were loaded or not. To display any CIMR components that were not loaded, enter **ZIMAG DISP IMAGE TPF02**. This will show you the information about all of the CIMR components for TPF02. If a CIMR component is missing, you can copy it from TPF01 using the ZIMAG COPY command (if no changes were made to that component). To display any IPL components that were not loaded, enter **ZIMAG DISP IPL** and look at the IPL2 line. It should show that both IPLA and IPLB are loaded. If an IPL area is missing, you must use the ZIMAG COPY IPL command or do an auxiliary load (TLDR) to load the missing area.

Moving Keypoints to the Working Area

Before Beginning

If the existing keypoints are compatible with the new image, you do not have to load or move new keypoints.

Because it is dangerous to overlay keypoints, they are not loaded directly to an image, they are loaded to a staging area. Use the ZIMAG KEYPT MOVE command to move the keypoints from the staging area (#KSAX) to the working area (#KEYPT). The ZIMAG KEYPT MOVE command prompts you to continue (ZIMAG KEYPT CONT) or abort (ZIMAG KEYPT ABORT). After you move the keypoints to the working area, perform a hard IPL to put them in core storage.

Note: Keypoints are shared for all images. If there is something wrong with a keypoint, that prevents you from bringing a system to NORM, IPL your loader general file and use the ZIMAG KEYPT RESTORE command to restore the keypoints you had before you entered the ZIMAG KEYPT MOVE command. This allows you to come up again on your prime mod.

IPLing an Image

To use an enabled image, you must perform a hard IPL, choose image selection, and enter the name of the image.

Notes:

1. If you are IPLing an image that has a different core layout from the previously used image, IPL with CLEAR to clear the VFA buffers.
2. If the image does not IPL successfully, IPL another image and debug the problem image.

Loading E-Type Programs to an Enabled System

Unlike the general file loader or the auxiliary loader, the E-type loader does not require an IPL.

The E-type loader:

- Does not require you to cycle down the system.
- Allows an unlimited number of programs to be loaded from a loadset.

Note: *Loadsets* are groups of programs loaded by the E-type loader. Each loadset is identified by a 5–8 character-unique name. This name is specified in the load deck used in the offline OLDR run. All programs in a loadset are activated or deactivated at the same time. The number of programs that can be associated with each loadset and the number of loadsets that can exist is limited only by the number of 4KB fixed file records (#OLDx) specified by you in the SIP RAMFIL macro (see *TPF System Generation*). Fixed file records are used to hold the programs loaded and the various tables necessary for the load. This allows for a dynamic load. Loadsets are processor shared, subsystem unique, and image unique.

- Supports loading of C load modules.

If you load a new TARGET(TPF) C library function, you must perform a C000 load and an online module IPL to rebuild the quick enter directory segment name table. Otherwise, you will not be able to use the new TARGET(TPF) C library function. If you replace an existing TARGET(TPF) C library function, a C000 load is not necessary.

Perform the following steps to load new E-type programs to a TPF system:

1. Create an E-type loader load deck.
2. Load the E-type programs to a storage medium.
3. Load a loadset of E-type programs from a storage medium to the system.

Creating an E-Type Loader Load Deck

Enter the JCL Cards

The following JCL cards are used to run the E-type loader offline segment (OLDR); *nn* is the release identification, and *ssid* is the subsystem ID.


```

//OLDRGDS JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A,REGION=2100K,
//      USER=mvsuser,PASSWORD=password
//NAME EXEC PGM=TPFLDRnn,REGION=9000K,PARM='OLDR,CLMSIZE=9000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//      DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//      DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//OBJLIB DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//CPRTMP DD UNIT=SYSDA,
//      DSN=&&CPRTMP,SPACE=(TRK,(100,20)),
//      DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//      DISP=(NEW,DELETE)
//PROGTEMP DD UNIT=SYSDA,
//      DSN=&&PRTEMP,SPACE=(TRK,(100,20)),
//      DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//      DISP=(NEW,DELETE)
//OUTPUT DD UNIT=SYSDA,VOLUME=SER=dasd,
//      DSN=mvsuser.OUTPUT,SPACE=(TRK,(40,20)),
//      DCB=(RECFM=F,BLKSIZE=4095,LRECL=4095),
//      DISP=(OLD,KEEP,KEEP)
//SALTB DD DSN=ACP.SALTB.LINK.RELnn.ssid,DISP=SHR
//ADATIN DD DSN=xxx
//ADATATMP DD DSN=&&SYSUT1,SPACE=(TRK,(100,20)),UNIT=SYSDA
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//PRINTER DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//LDRTRACE DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE

```

Figure 25. OLDR Run Load Deck (To GDS) Example

Notes:

1. The DD names SALTB, OBJLIB, OUTPUT, PRINTER, and SYSIN are required.
2. DD name LDRTRACE is required only if you specify TRACE(ON) as a parameter.
3. Data set concatenation is sequence dependent.
4. Figure 25 shows an OLDR load deck using a GDS as the storage medium. "JCL Load Deck Examples" on page 417 shows JCL load decks using other storage medium.
5. If the storage medium is tape, use standard label tapes.

TPFLDR Exec Card: The TPFLDR EXEC card identifies the specified version of the TPF offline loader program (TPFLDR).

These are the parameters that can be specified on the EXEC card.

1. OLDR
indicates the E-type file loader program (OLDR) is to be executed.
2. CLMSIZE = x
where x is greater than or equal to the MVS file size of the largest C load module to be loaded. If a C load module contains a higher than normal ratio of VCONs to executable code, CLMSIZE must be larger.
3. ADATASIZE = x
where x is greater than or equal to the MVS file size of the largest ADATA file to be loaded. ADATA does not attempt to load ADATA files when ADATASIZE=0.
4. TRACE(ON|OFF)

If you specify TRACE(ON), selective trace data will be written to the LDRTRACE DD name. This trace data can be useful to system programmers to debug problems that occur while running TPFLDR. TRACE(ON) is primarily intended for TPF development personnel for this purpose. TRACE(OFF) is the default. If you specify TRACE(OFF), no trace data is written.

STEPLIB Card: The STEPLIB card identifies the partitioned data sets that contain the offline loader programs.

LOADMOD Card: The LOADMOD card identifies the partitioned data set (LOADMOD) that contains the link-edited C load modules. When an E-type program is being loaded, the data sets identified by the LOADMOD card are searched before the data sets identified by the OBJLIB card are searched. The C load modules are compiled and link-edited using standard MVS facilities. For more information, see *TPF System Generation*.

OBJLIB Card: The OBJLIB card identifies object libraries. Object libraries are partitioned data sets of assembled or compiled TARGET(TPF) programs. All programs to be loaded, with the exception of C load modules, must be in object libraries. You cannot load the control program and FACE table using the E-type loader. Library contents are maintained and updated using MVS procedures. For more information see *TPF System Generation*.

CPRTEMP Card: The CPRTEMP card creates temporary data sets that contain compacted program allocation table (PAT) records.

PROGTEMP Card: The CPRTEMP card creates temporary data sets that contain the program records associated with a loadset.

OUTPUT Card: The OUTPUT card identifies the output device to which offline programs and data are written.

SALTB Card: The SALTB card identifies partitioned data sets that contain the system allocator table (SALTBL). The SALTBL contains information necessary for the TPF linkage editor to resolve virtual address constants (VCONs). These VCONs can be program addresses, tape names, macro parameters, and others that are put in the SALTBL to allow relocation of resources in the online system. See "System Allocator" on page 399 for more information on the SALTBL.

ADATAIN Card: The optional ADATAIN card identifies the partitioned data set (PDS) that contains ADATA files to be loaded with real-time programs. If the ADATAIN card is present, OLDR will attempt to find ADATA files in the data set specified by the ADATAIN card. SYSADATA files are generated by the high-level assembler (HLASM) and must be processed by the SYSADATA postprocessor (program TPFDBG) to create ADATA files before they can be loaded. When OLDR loads a basic assembly language (BAL) program, it will attempt to find an ADATA file by searching for a PDS member with a matching name. When OLDR loads an E-type load module, it will attempt to find an ADATA file for each object module linked into the load module. OLDR will ensure that the assembly date of the ADATA file matches the assembly date of the assembler language program before loading an ADATA file.

Note: If a program was assembled more than once on a given date, it is possible for OLDR to load an incorrect ADATA file with an assembler language program. You must ensure this does not happen. Whenever an E-type program (BAL or load module) is loaded to the online system and accepted as a new base version by the E-type loader, any ADATA files

associated with a previously loaded base version of that program is effectively erased. For more information, see *TPF System Generation*.

ADATATMP Card: The ADATATMP card specifies a temporary data set used to hold ADATA files before they are written to the output medium.

PRINTER Card: The PRINTER card produces an output listing that details all programs loaded to the storage medium. Errors that do not cause OLDR to abend are also written in this listing. Abend error messages and informational messages are sent to the operator console. For more information about error messages, see *Messages (System Error and Offline)* and *Messages (Online)*.

Enter the Input Control Cards

The following control cards are presented in the order in which they should be placed in the load deck. This order is not affected by the addition of comment cards.

Comment Card: Comment cards can be placed anywhere in the load deck. Comments are identified by an asterisk (*) in card column 1.

Subsystem ID Card:

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

SYSID=nnnn

nnnn = Subsystem ID

The Subsystem ID card must be the first noncomment card in the deck. It identifies the subsystem to be loaded. This ID is also the prefix of the command that activates the online segment.

Program Allocation Table Card:

Program Allocation Table Card (Format 1)

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

PATVERS=vv

vv = version number

Program Allocation Table Card (Format 2)

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

PATVERS=TIME

Program Allocation Table Card (Format 3)

|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8

PATVERS=NONE

The Program Allocation Table card tells the E-type loader to compare the specified version of the offline IPAT with the specified version of the online IPAT. If they are identical, processing continues. If they are not identical (they were not created at the same time), a check is made to determine if they are compatible. They are considered compatible if new programs in the offline version replaced spare slots in the online version, obsolete programs were changed to spare slots in the offline version instead of deleting the entry in the allocator deck, and all programs appear at the same relative location in both the offline and online version. If the 2 versions are compatible, but the offline version contains newly allocated programs, the

E-type loader changes the online version to contain the new programs. If the versions are not compatible, a message tells you why they are not and the load ends.

You can use this feature to replace spare PAT slots with allocation information for newly added programs.

Notes:

1. If you do not want to change the online IPAT or check the compatibility of the online and the offline versions of the IPAT during a ZOLDR LOAD, then the Program Allocation Table card must contain NONE in columns 9–13.
2. If you want to check that the offline PAT is identical to the online IPAT by ensuring their time stamps are identical during a ZOLDR LOAD, the Program Allocation Table card must contain TIME in columns 9–13. If the time stamps are not identical, the load ends.

System Allocator Card:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
```

```
SALVERS=vv
```

vv = version

The System Allocator card tells the TPF linkage editor which version of the SALTBL to use for link-editing.

Loadset Card:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
```

```
LOADER LOADSET nnnnnnnn
```

nnnnnnnn = 5 to 8-character loadset name

The Loadset card puts the Call Program cards that follow it into a group of programs called a loadset. This loadset of programs is identified by the loadset name on the Loadset card. These loadsets are used to enter programs into a running TPF system.

Note: There can be more than one Loadset card per load deck.

Call Program Card:

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
```

```
LOADER CALL PROG nnnnvv
```

Alternate Call Program Card

```
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8
```

```
nnnnvv
```

nnnn = program name

vv = version number

One or more Call Program cards must follow the Loadset card. Call Program cards tell the offline segment which programs are to be loaded to storage mediums.

Patch Cards: Any REP cards for a particular program must follow the Call Program card for that program. Columns 75–80 must match columns 1–6 of the Call Program card (or columns 21–26 if the alternate format is used). The E-type loader permits a maximum of 255 patches per program.

REP cards are not supported for C load modules.

Note: If you include a REP card to patch a real time program that will have an ADATA file loaded, the TPF Assembler Debugger for VisualAge Client may not display an accurate listing view of the program.

Loading E-Type Programs to a Storage Medium

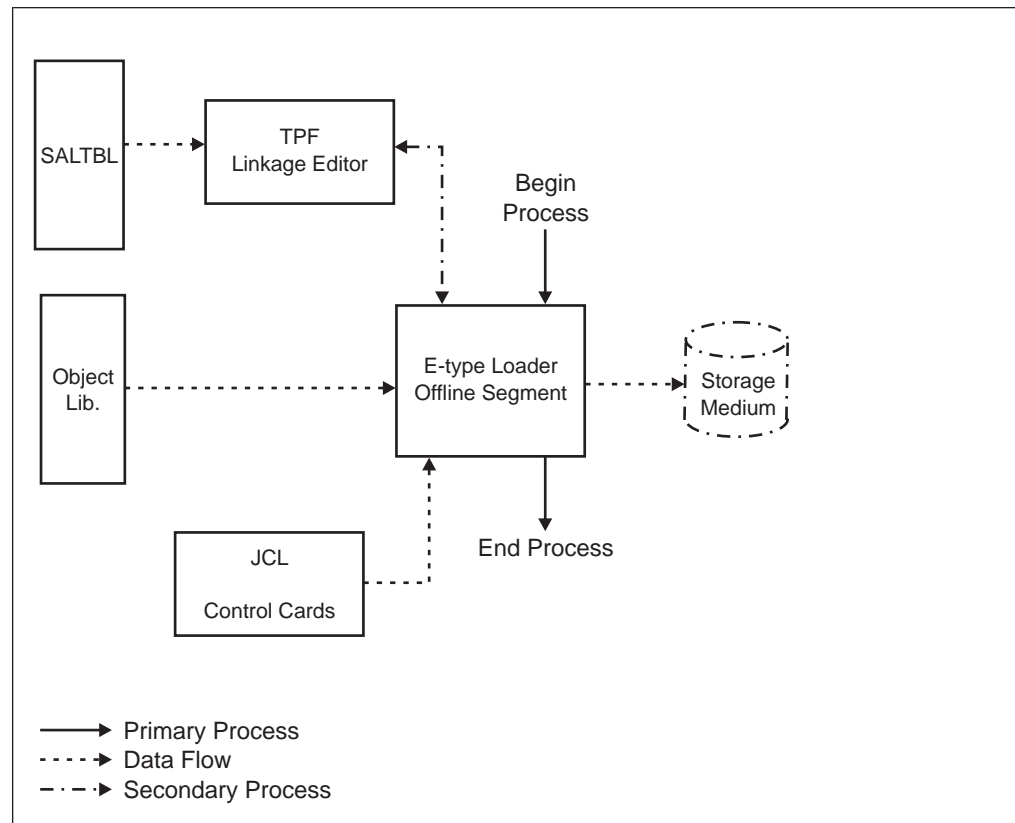


Figure 26. E-Type Load via E-Type Loader (Offline — MVS)

The E-type loader offline segment runs under MVS control and places loadsets of programs (identified by the input control cards) on a storage medium. The storage medium is then loaded to the system by the ZOLDR command.

Note: If you are writing to a virtual reader, use the ELDRVRDR EXEC to convert OLDR JOB output to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

Loading and Activating a Loadset of Programs

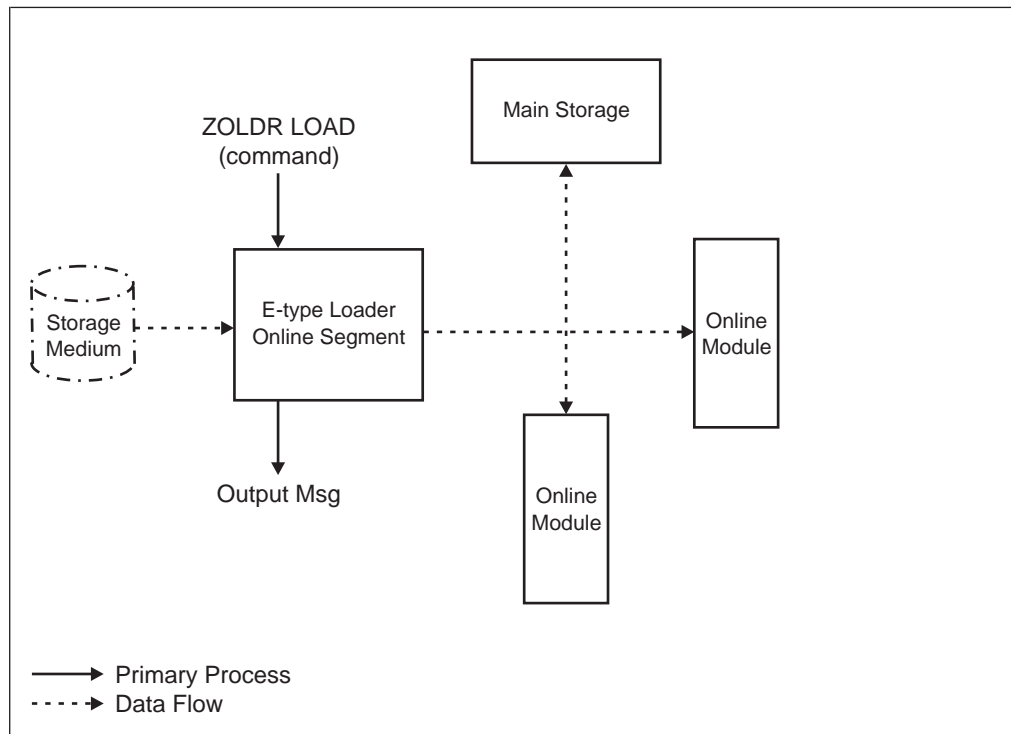


Figure 27. E-Type Load via E-Type Loader (Online — TPF)

If you want to replace a base version of an E-type program, change the program in main storage using the ZAPGM command or load and activate a loadset of programs. Figure 27 shows how a loadset of programs are loaded from a storage medium to the system using the ZOLDR LOAD command.

Note: ZAPGM should not be used for major program changes because it permits you to change only 16 bytes of a program. Also, ZAPGM is *not* supported for C load modules. ZAPGM can be used to make an immediate change to a program that is causing system problems. In most cases, make program changes by loading and activating a loadset of programs.

See *TPF Operations* for more information about the ZAPGM command.

Once you have loaded an E-type loader load deck to a storage medium, load and activate the loadset of programs by doing the following:

1. Load a loadset of programs from the storage medium to the system.
2. Allocate programs that are unallocated (required to accept programs as base versions and to run programs with transfer vectors).
3. Activate the loadset and test the new programs.
4. Accept the loadset (if you want to replace the base version programs with the new versions).

Loading a Loadset of Programs

Use the ZOLDR LOAD command to load a loadset of programs. ZOLDR LOAD reads program records from an input device and writes them to 4KB fixed file records.

Note: If you are loading from a virtual reader, make sure your OLDR job output has been converted. Use the ELDRVRDR EXEC to convert OLDR JOB output to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

The names of the loaded loadsets are maintained in loadset directory (LSD) records. The names and temporary file locations of the loaded programs are maintained in E-type loader program directory (EPD) records. The versions of the loaded programs are maintained in program version records (PVRs). Loaded programs have no effect on system activity until they are activated.

Allocate Programs That are Unallocated

Programs must be allocated before they can be accepted as the base version. Programs that have transfer vectors must be base allocated or you will receive a CTL-063 condition when you try to run the transfer vector program.

Allocating new programs in an active TPF operating system can be done without an IPL by doing the following:

1. Change the spare entries in your allocator deck to the unallocated programs you want to allocate.
2. Run SALO with the updated allocator deck to create an updated offline allocator with a new Program Allocation Table (IPAT).
3. Create an E-type loader load deck with the updated allocator version on the PATVERS card.
4. Run the E-type loader offline segment to load the load deck to an intermediary device.
5. Enter **ZOLDR LOAD *ddname* PATU** (where *ddname* is the data definition of the intermediary device) to update the online PAT on all processors. The previously unallocated programs are now allocated.

Note: See “System Allocator” on page 399 for more information about allocating programs and transfer vectors.

Activating a Loadset to Test New Programs

Use the ZOLDR ACTIVATE command to activate a loadset of programs. ZOLDR ACTIVATE causes programs in a loadset to become available for processing. You can activate a loadset of programs on one or all processors.

Selectively Activating E-Type Programs

You can control the use of E-type programs by selectively activating a loadset of programs. Use the SEL option of the ZOLDR ACTIVATE command to limit the use of new programs to specific terminals, lines, users, and others.

Note: Use the selective activate user exits to specify the terminals, lines, users, and others that you want to allow to use a specific loadset of programs. “Selective Activate Exits” on page 109 provides information to help you write the code needed to support the SEL option.

Using Loadsets

When using loadsets, use a ZOLDR command to do the following:

1. Exclude a program from a loadset of programs.
2. Reinclude a program to a loadset of programs.
3. Display loadset information.

4. Deactivate a loadset of E-type programs.
5. Delete a loadset of E-type programs.
6. Reclaim system resources.
7. Change E-type loader program allocation characteristics.
8. Change E-type loader values.
9. Clear E-type loader file resident records.

You may want to use the ZDEAT command to display ECB status.

Note: For online help, type **ZOLDR ?**, **ZDEAT ?**, or see *TPF Operations* for more information.

Excluding a Program from a Loadset

Use the ZOLDR EXCLUDE command to exclude a program from an existing loadset of programs. If a loadset is active, ZOLDR EXCLUDE deactivates the program on all processors. A program that is excluded from a loadset of programs will not be included in later functions performed on the loadset. For example, if you exclude the PRGM1 program from the GROUP1 loadset, PRGM1 is not activated when you activate GROUP1.

Reincluding a Program to a Loadset

Use the ZOLDR REINCLUDE command to reinclude a program to an existing loadset of programs. A program that is reincluded to a loadset of programs will be included in later functions performed on the loadset. For example, if you reinclude the program PRGM1 to loadset GROUP1 then PRGM1 (which had previously been excluded from GROUP1) is activated when you activate GROUP1.

You must deactivate an active loadset before an excluded program can be reincluded into the loadset.

Displaying Loadset Information

Use the ZOLDR DISPLAY command to display the following information:

- Status or contents of a loadset
- Intersections with other loadsets
- Active loadsets
- All loadsets containing a particular program
- All loadsets
- E-type loader values and rules.

Deactivating a Loadset of E-Type Programs

Use the ZOLDR DEACTIVATE command to deactivate a loadset of programs. ZOLDR DEACTIVATE causes all programs in a loadset to become inactive when they finish processing. When all programs in a loadset are deactivated, previously active versions of the programs are used by new entries.

Note: The FORCE option processes before any previously scheduled ZOLDR tasks. It should only be used if ZOLDR DEACTIVATE processing ends with an error. The FORCE option can change the version of a program that is currently in use by an entry. This could cause interface problems.

Deleting a Loadset of E-Type Programs

Use the ZOLDR DELETE command to delete a loadset of programs. ZOLDR DELETE causes all of the fixed file records associated with a loadset of programs to be returned to the pool of available fixed file records. ZOLDR DELETE cannot delete an active loadset until it has been deactivated.

If a program in the loadset is still active, the system changes the loadset name to *-nnnn* (where *nnnn* is 0000–9999) and marks it as *delete pending*. The system deletes the renamed loadset when all ECBs using the loadset are finished.

Reclaiming System Resources

Use the ZOLDR RECLAIM command to make inaccessible fixed file records available to the system.

If an E-type loader action is interrupted by a system error, some fixed file records may not be able to be accessed by the system. Although the information in the records is still available elsewhere, the original fixed file record may not be accessible. If the fixed file record is not accessible, the system cannot use that disk space. Therefore, you must enter ZOLDR RECLAIM to allow the system to be able to use that disk space again.

Note: Your site's procedures will determine how you use the ZOLDR RECLAIM command. You may want to enter ZOLDR RECLAIM after any E-type loader RESTART, or you may want to enter ZOLDR RECLAIM every night to reclaim inaccessible disk space.

Changing E-Type Loader Program Allocation Characteristics

Use the ZOLDR ALTER PROGCHAR command to alter the allocation characteristics of unallocated E-type programs.

Changing E-Type Loader Threshold Values

Use the ZOLDR ALTER command to alter the following E-type loader threshold values:

- Extra program allocation table (PAT) slot threshold percentage
- E-type loader fixed file record threshold percentage
- Number of incompatible PAT slots to report during the ZOLDR LOAD function before the process is aborted
- Time interval for starting the E-type loader police routine
- Time interval for starting the E-type loader long running job detection routine and the E-type loader reclaim detection routine.

Clearing E-Type Loader File Resident Records

Use the ZOLDR CLEAR command to clear and initialize all file resident records the first time the system is IPLed or if E-type loader records are damaged.

Displaying ECB Status

Use the ZDEAT commands to see the number of ECBs using each activation number and if the activation number corresponds to a selectively activated loadset.

Accepting a Loadset of E-Type Programs

Before Accepting a Loadset

Issue the ZOLDR ACCEPT command only if you are satisfied that the programs in the loadset work correctly and you want them online. If you are still testing the loadset of programs, you likely want to keep them activated only.

ZOLDR ACCEPT permanently replaces the base versions of the programs with an active loadset of programs. If you make a mistake, reload the base versions of programs, activate them, and accept them to correct your mistake.

Unallocated programs must first be allocated before you can accept them.

Use the ZOLDR ACCEPT command to accept a loadset of programs. ZOLDR ACCEPT causes the active loadset of programs to overlay existing programs at base allocated addresses. The fixed file records associated with an accepted loadset of programs are deleted after you perform the accept.

Notes:

1. The loadset must be activated on all processors to be accepted.
2. A new program will not be accepted if there is an ECB on any subsystem that can still use the base version program.
3. If active loadsets intersect, you must accept them in the order they were activated. That is, the first activated loadset will be the first accepted loadset.
4. Use the ZOLDR DISPLAY command before accepting a loadset of programs to make sure that you are accepting the correct version of a program.

Using E-Type Loader Functions

E-type loader functions affect only the program base that is being used on the processor from which the function was requested. If other processors are using a different program base, those processors do not see the effects of E-type loader functions until they begin using the same program base.

For example, in a loosely coupled environment where processor A is using program base 1 and processor B is using program base 2, if you issue the ZOLDR ACTIVATE command on processor A, the specified loadset is started on all processors using program base 1. Because processor B is using program base 2, the specified loadset is not actually activated on that processor until you perform an initial program load (IPL) using program base 1.

Record ID Attribute Table

The record ID attribute table (RIAT) is used to describe the characteristics of both fixed and pool file records. Characteristics of fixed file IDs include: exception recording, logging, and restoring status, user exit status, VFA candidacy, locking status, and record caching candidacy. These characteristics also apply to pool file IDs. Pool file IDs have these additional characteristics: size, duration, duplication status, and device type. These characteristics can be specified for any of the 10 record categories (labeled 0–9) for a specific record ID.

The 10 pool record categories are available to application programmers for more efficient utilization of disk space. Each record category should have unique attributes for size, pool, and device. For example, record category 0 (RTP0) could be designated as size=L (large), pool=LT (long term, nonduplicated), and device=A (DEVA). No other record category (RTP1 — RTP9) would have these same attributes.

Every RIAT-controlled record has a 2-byte record ID, a 16-bit combination ranging from 256 to 64 511, associated with it. The TPF system reserves the record IDs in the range 1–255 and 64 512–65 535, for its own use, and record ID 0 is considered to be invalid.

In an MDBF environment each subsystem requires its own RIAT.

If multiple TPF images are defined, each image can have its own unique RIAT.

Contents of the RIAT

RIAT data begins at the beginning of the table area; there is no conventional TPF header. RIAT entries are mapped by the DCTRIT data macro.

The RIAT is divided into 2 parts. The first part contains 4-byte hash pointers into the second part of the table. The number of hash pointers in the first part of the RIAT is determined by the number of RIAT entries and is computed in the RIATA macro.

The second part of the RIAT table consists of sequential entries, one per RIAT ID. Each entry consists of attribute information, the record ID itself, and a synonym chain pointer, if necessary.

The attribute information for each ID consists of the following data:

- Record category (0 — 9)
- Exception recording, logging, and restoring data
- User exit data
- VFA data
- Record caching candidacy
- Lock maintenance in a loosely coupled environment.

The synonym chain pointer is an address pointer that is set up if 2 different record IDs hash to the same value. By following the chain of pointers, the desired record ID can be retrieved.

Addressing the RIAT

For each ID from which data is to be retrieved, a hash value is first computed by dividing the desired record ID by the size of the hash table and saving the remainder. By multiplying this remainder by 4, the correct pointer into the second part of the RIAT table is accessed.

Once the second part of the RIAT table is accessed, the desired ID is compared against the ID corresponding to the RIAT entry currently being accessed. If these IDs match, the correct entry has been found. If the IDs do not match, the synonym chain pointer is used to access another RIAT entry. By continuing to follow this scheme, the correct entry is eventually located.

Programming Areas

1. IPLB

IPLB loads the RIAT into storage along with all the other main storage resident chained tables (FCTB, IPAT, SIGT, and others).

2. CTIN

CTIN places the address of each subsystem's RIAT in the CINFC area of the subsystem in the slot addressed by CINFC tag CMMRIT.

CTIN also builds the hash table in the first part of the RIAT and sets up all addresses in both parts of the RIAT.

Programming Techniques

The ZRTDM command is used to display or modify the RIAT online. The RIAT is directly accessed only through the RITID macro.

Record Size

The size of each RIAT depends on the number of entries coded with RIATA macro calls.

Frequency of Access

The RIAT is always loaded into main storage. The RIAT file copy is accessed when a specific entry is to be updated using the ZRTDM command.

Record Life

The RIAT exists for the life of the system. A new RIAT can be generated and loaded with the general file or auxiliary loaders at any time.

Record Generation

To locate the RIAT, SIP Stage I retrieves the data set name (coded in the INDSN macro) and the member name in which the RIAT resides from SPPGML. The RIAT, before assembly, is made up of RIATA statements beginning with RIATA START and ending with RIATA FINISH. During SIP Stage II, the RIAT is put in an offline program list, assembled, and loaded.

Optional parameters (defaulted if not coded) include:

- Record category (0 — 9) characteristics
- Exception recording, logging, and restoring data

- User exit data
- VFA data
- Record caching candidacy
- Lock maintenance in a loosely coupled environment.

References

- See *TPF System Generation* for more information about the SRIAT and RIATA macros.
- See *TPF Operations* for more information about the ZRTDM command.

Multiple Assembly/Compilation Print Program

The Multiple Assembly/Compilation Print (ASMP) program allows one or more of the Multiple Assembly/Compilation program listings to be printed from tape or disk.

The listings that are read can use either ANSI (FBA) or Printer Channel Command (FBM) control characters. ASMP defaults to ANSI control characters, but if printer channel commands are desired, code PARM=M on the JCL EXEC statement.

Note: Although the High Level Assembler (ASMA90) produces listings in FBM format with a record length of 121, the MASM program converts the listings to FBA format with a record length of 133. Also, the C compiler listings are converted by MASM to FBA format with a record length of 133. This was done to allow assembler and compilation listings to be read from the same MVS volume.

Printing Multiple Assembly and Compilation Listings

The Multiple Assembly/Compilation Print (ASMP) program causes multiple assembly and compilation listings to be printed. Each listing will be preceded by a header page identifying the program name. You control what gets printed by changing the JCL control cards.

To print multiple assembly and compilation listings, make a JCL control deck and run the deck as a batch job under OS.

JCL Control Cards

The JCL control cards are an EXEC card for the program, a STEPLIB or JOBLIB card for the PDS in which the program is stored, and the following cards:

- | | |
|-----------------|--|
| MSGFILE | This DD card specifies a SYSOUT DD statement for messages from the program. |
| LISTAPE | This DD card specifies the input tape or disk created during a MASM run. |
| PRINTOUT | This DD statement specifies a SYSOUT DD statement for the listings that are being retrieved from tape. |
| ASMPCTL | <p>This card controls whether all listings or specific listings are printed.
//ASMPCTL DD DUMMY prints all of the listings on the input tape.
//ASMPCTL DD *, with the LIST parameter, prints the specific listings included in the LIST statement.</p> <p>LIST This card controls which program listings are printed. One or more program listings can be requested. If there is only one request in the list, parentheses are not required. The program name is the member name in the source PDS from which it was generated. The names are not required to be in any order.</p> |

Note: If a suffix was attached to the program name during the assembly or compilation, make sure it is included in your LIST statement

A list can appear on several cards. The following example shows a list continued on more than one card:

```
LIST=(PPCP40,,BMP040,BMP140,BMP240,BMP340,BMP440,
BMP540,BMP640,BMP740,BMGL40,LTPP40,LTPQ40,ACPL40,
ICDF40)
```

Note: If continued on the next card, the last name on the previous card must be followed by a comma. Only columns 1–71 of any card can be used.

The sample JCL that follows is designed to print 2 programs from tape.

```
//LISTJOB EXEC PGM=ASMPvv
//STEPLIB DD DSN=ACP.LINK,RELvv,DISP=SHR
//LISTAPE DD DSN=PRINT,UNIT=TAPE,VOL=SER=LISTAP,DISP=OLD,
// DCB=(BLKSIZE=23940,RECFM=FBA)
//PRINTOUT DD SYSOUT=A,DCB=(BLKSIZE=133,RECFM=FBA)
//MSGFILE DD SYSOUT=A,DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//ASMPCTL DD *
LIST=(CZXPA0,CZXDA0)
/*
```

Notes:

1. vv refers to the actual TPF release number being executed against.
2. The LISTAPE data definition must define as input the same device (for example, tape or disk) as the LISTAPE data definition that was used to execute MASM. Also, if you have more than 1 input tape, specify all of the tape IDs (for example, VOL=SER=(123456,789012)).

Return Codes

- | | |
|---|---|
| 0 | Successful execution. |
| 4 | One or more programs specified in LIST option not found (see error messages). |
| 8 | Error in ASMPCTL data definition (see error messages). |

Error Messages

- | | |
|---------------------|---|
| Message: | KEYWORD ON CONTROL CARD IS NOT -LIST- |
| Explanation: | A control card has been read on which the keyword is not LIST. |
| Message: | INSUFFICIENT CORE - INCREASE REGION SIZE |
| Explanation: | The program issued a GETMAIN request for core to build a list from the control cards. The GETMAIN was unsuccessful. The program should be executed with a larger region or partition size. |
| Message: | MAX (n) NO OF ENTRIES ALLOWED IN CONTROL CARD LIST EXCEEDED-INCREASE VARIABLE &LISTMEM AND REASSEMBLE |
| Explanation: | The program can print a list containing as many as <i>n</i> programs. The list that is input to the program exceeds this number of entries. This can be resolved by changing the value of local variable &LISTMEM in the program and reassembling the program. An easier way to resolve this problem might be to run the program more than once with a smaller control card list. |

Message: INVALID SYNTAX IN CONTROL CARD

Explanation: Self-explanatory.

Message: ENTRY IN CONTROL CARD LIST EXCEEDS 8 CHARACTERS

Explanation: Self-explanatory.

Message:

***** ASMP ERROR REPORT *****

-----NOT FOUND ON LISTAPE

Explanation: Program identified was requested via LIST option but not found during search of LISTAPE input. Verify program name from output of MASM.

Hardware Requirements

Hardware and software are required which will be capable of running the program under OS and which has a tape or DASD drive on which to input the print tape or DASD.

Macro Cross-Reference

The macro cross-reference listing is generated through execution of the macro cross-reference programs DCRS and DREF with the Sort/Merge program and a single MVS library containing card image format.

The DCRS cross-reference program is designed to scan assembler language members of partitioned data sets. PL/I program names are checked for exclusion from the report. Scanning macro or C language source code members will cause unpredictable results.

DCRS searches every card image from each member of a partitioned data set. You specify the search targets, which can include macros, global fields, system equates, CINFC fields, CONKC fields, tape tags, CZ1CP fields, SYCON fields, and user-specified strings (also called user tags). When a match is found, the program name, macro name, field, or user tag is placed in a work area. Next, the work area is passed to the Sort/Merge program, which arranges the data under the designations specified by DCRS. Finally, a sorted copy of the work area is passed to the DREF program. DREF formats the information for the printer and produces headings for each search category. The headings for the printed output from the DREF program include the date of compilation. The JULTOACT program converts the Julian calendar date to the actual date, if the SET DATE information has been properly entered.

Specifying the DCRS Search Parameters

You can specify the DCRS searching option with the PARM keyword parameter of the JCL EXEC statement that invokes DCRS. The format of the PARM parameter is:

PARM=*'[/size/]option'*

/size/

The size of the ICALL translation table. The *size* must be a 4- or 5-digit decimal number. If this parameter is omitted or not correct, DCRS uses the default value of 19 584 bytes.

The ICALL translation table is built by DCRS and used to map the symbolic name of a TPF Advanced Program-to-Program Communications (TPF/APPC) segment to the TPF segment name. (ICALL is an internal macro used in the TPF/APPC support code. See *TPF System Macros* for additional information about this macro.)

The size of the table is determined by multiplying the number of segments that can be called with the ICALL macro (that is, they have a symbolic name) by the maximum size of each table entry. Each entry in the table can be a maximum of 68 bytes long; 64 bytes for the symbolic name plus 4 bytes for the TPF segment name.

Note: An example of this option is shown in "Example of DCRS" on page 382.

option

The search option that DCRS will use. Use one of the following options:

- 1** Macros, system equates, globals, SYCON and CZ1CP fields
- 2** Macros only

- 3 System equates and global fields only
- 4 SYCON and CZ1CP fields only
- 5 Macros, system equates, and global fields
- 6 Macros, SYCON and CZ1CP fields only
- 7 System equates, global, SYCON, and CZ1CP fields
- 8 CONKC and CINFC fields
- 9 Tape macros and labels

A, tag1, tag2, ...

User-specified tags to be used as search arguments. The format for this option is:

PARM='A,TAG1,TAG2,...,TAGn'

For example, if you specify:

'A,CLRIO,SIOF,HIO,SCK'

DCRS will compare all opcodes and operands with the strings CLRIO, SIOF, HIO, and SCK.

Notes:

1. Any occurrence of the user tag in an opcode or operand is recognized as a match. When a match is found, the user tag and the segment name are placed in the work area.
2. The maximum number of total characters for the PARM list is 100. Each individual tag must be 8 characters or less, and contain only alphabetic, numeric, #, or @ characters.
3. If the PARM parameter is omitted or not correct, the default is PARM=1 (macros, system equates, globals, SYCON and CZ1CP fields).
4. Options 1, 2, 5 and 6 will only locate macros with names that are 5–8 characters long. Use the A option to locate macros that are not 5–8 characters long.

Example of DCRS

The following is example of the DCRS PARM parameter using the size option:

PARM='/1234/1'

This statement sets the size of the ICALL translation table to 1234, and searches macros, system equates, globals, SYCON, and CZ1CP fields.

Specifying a DREF Heading Parameter

You can specify the DREF heading option with the PARM keyword parameter of the JCL EXEC statement that invokes DREF.

Using the PARM field, a heading can be specified for the report. The default heading is NO HEADING SPECIFIED. See “Control Cards” on page 383.

Input to the Macro Cross-Reference Programs

A macro cross-reference listing is generated by running the DCRS, SORT, and DREF programs.

DCRS Program

DCRS searches, according to specified search options, a single MVS library which must be a partitioned data set containing Basic Assembler Language (BAL) programs in card image format. PL/I program names are checked for exclusion from the report. Note that DCRS is not designed to read macro language or C language.

Three inputs are required by DCRS. The first is the scanning option which has already been discussed. Simply code the desired option onto the PARM parameter of the JCL EXEC statement. The second and third inputs describe the MVS PDS to be scanned. IN and INN are the input DD statements for DCRS. Both IN and INN should be coded with the same PDS name. The duplication is necessary because DCRS opens IN as a sequential data set, so that the directory blocks can be read. INN is opened as a partitioned data set so that the members contents can be read.

SORT Program

The sort program reads in the output data set produced by the DCRS program. Code DCRS's output data set as input on the SORTIN DD statement. On the SYSIN DD statement, pass the instructions that will sort the cards by character, in ascending order, on columns 1–25. The character in column 1 is assumed to be the DCRS scanning category.

DREF Program

The DREF has 2 inputs. First, an optional header can be included in the formatted output. Code the PARM parameter of the JCL EXEC statement with the desired string (as many as 100 characters). The second input is the temporary sequential data set created by the sort program, which contains alphabetically sorted records. Code the name of the data set on the IN DD statement.

Control Cards

The following control cards are an example of the cards necessary to execute the macro cross-reference programs under MVS:

```
//DCRSTST JOB      MSGLEVEL=1
//DCRS1   EXEC     PGM=DCRSvv,PARM='1'
//STEPLIB DD       DSN=ACP.LINK.RELvV.BSS,DISP=SHR
//SYSUDUMP DD      SYSOUT=A
//ERR      DD      SYSOUT=A
//IN       DD      DSN=ACP.SRCE.RT1.RELvV,DISP=SHR
//INN      DD      DSN=ACP.SRCE.RT1.RELvV,DISP=SHR
//ICALLDD  DD      DSN=ACP.MACRO.RELvV.BSS(ICALL),DISP=SHR
//OUT      DD      DSN=&&SORTIN,UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(CYL,(10,8)),
//          DCB=(RECFM=FB,LRECL=25,BLKSIZE=4000)
//DCRS2   EXEC     PGM=SORT
//SYSOUT   DD      SYSOUT=A
//SYSPRINT DD      SYSOUT=A
//SORTLIB  DD      DSN=SYS1.SORTLIB,DISP=SHR
//SORTIN   DD      DSN=&&SORTIN,DISP=(OLD,DELETE),UNIT=SYSDA
//SORTOUT  DD      DSN=&&SORTOUT,UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(CYL,(20,8)),
//          DCB=(RECFM=F,LRECL=25,BLKSIZE=25)
//SORTWK01 DD      UNIT=SYSDA,SPACE=(CYL,8,,CONTIG)
//SORTWK02 DD      UNIT=SYSDA,SPACE=(CYL,8,,CONTIG)
//SORTWK03 DD      UNIT=SYSDA,SPACE=(CYL,8,,CONTIG)
//SYSIN    DD      *
SORT      FIELDS=(1,25,CH,A)
/*
//DCRS3   EXEC     PGM=DREFvv,PARM='THIS IS THE OPTIONAL HEADING'
//STEPLIB DD      DSN=ACP.LINK.RELvV.BSS,DISP=SHR
```

```
//IN      DD      DSN=SSORTOUT,UNIT=SYSDA,DISP=(OLD,PASS)
//OUT     DD      SYSOUT=A
/*
//
```

where *vv* is the correct version ID.

Procedure

Use the following instructions to run the macro cross-reference programs to produce a printed cross-reference report.

1. Mount and ready all packs containing input library and work areas.
2. Ready the printer.
3. Start the reader (MVS command).

Output from the Macro Cross-Reference Programs

Output from the macro cross-reference programs are in the form of listings or files.

Listings

1. DCRS
The ERR data set will contain any error messages or warnings issued by DCRS. See "DCRS Attention Messages" for a complete description of the messages.
If there are no errors scanning the library, DCRS does not issue listings or messages.
2. DREF
Printing of the macro cross-reference listing, containing as many as 11 sections with headings.

Files

1. DCRS
Temporary sequential data set to be sorted by an MVS sort. This data set must be defined by the OUT DD statement with a block size of 4000 fixed block length 25 records. For example, code the DCB parameter as:
DCB=(RECFM=FB,LRECL=25,BLKSIZE=4000)
2. SORT
Temporary sequential data set, alphabetically sorted. This data set must be defined by the SORTOUT DD statement with unblocked length 25 records. For example, code the DCB parameter as:
DCB=(RECFM=F,LRECL=25,BLKSIZE=25).

DCRS Attention Messages

For DCRS attention messages, the card image, the segment name, and a reason are moved into the ERR data set, followed by one of the following reasons:

ICALL PARAMETER *parm* NOT FOUND IN ICALL MACRO

Explanation: This message can be caused by one of the following conditions:

- The wrong copy of the ICALL macro was used
- The symbolic name to be translated is incorrect
- The translate table does not contain all the necessary entries.

INVALID PARM DATA. DEFAULTS USED FOR STORAGE AMOUNT AND OPTION.

Explanation: The data specified on the PARM parameter is invalid. The function continues using the default storage amount of 19,584 bytes and the default option of 1.

OPEN FAILURE ON DDNAME ICALLDD

Explanation: The open failed on the DDNAME used to access the ICALL macro. ICALL macros do not appear in the listing.

OPERANDS TRUNCATED

Explanation: The buffer that holds the operands is filled. The rest of the operand string is ignored. The

DCRS Error Messages

For DCRS error messages, the card image and segment name are moved into the ERR data set. Possible reasons are:

BLANK LINE FOUND

Explanation: DCRS has found a card image that contains only blanks. The card image is not a continuation of the previous line.

COMMA BEFORE INST

Explanation: A comma was found preceding before a valid label or opcode. (INST is an abbreviation for instruction.)

INVALID LABEL

Explanation: The instruction's label contains invalid characters.

The following error messages occur without the card image and segment name being displayed:

ABEND 9

Explanation: The data set coded for the ERR DD statement cannot be opened. DCRS issues an ABEND with a user return code of 9. The job is terminated and a dump of virtual storage areas relevant to the job is printed if a SYSABEND, SYSMDUMP or SYSUDUMP DD statement is provided in the JCL for this ABENDING job.

A DATA FILE WAS UNSUCCESSFULLY OPENED

Explanation: An unsuccessful attempt to open an input or output data set, except for the data set defined for the ERR DD statement, causes the job to end and a message to print.

card image that is displayed will be the last card image of the continued BAL statement. The buffer holds a maximum of 255 characters.

STORAGE FOR ICALL TRANSLATE TABLE IS NOT LARGE ENOUGH

Explanation: The storage block size specified for the table is not large enough. Change the block size specified on the PARM parameter.

INVALID OPCODE

Explanation: The instruction's opcode contains invalid characters.

NO ENDING QUOTE

Explanation: After finding the start of a quoted string, DCRS did not find the closing apostrophe to end the quoted string.

PARSING ERROR

Explanation: DCRS's parser failed to recognize the instruction's format.

ERROR READING BLOCK FOR SEGMENT - segment name

Explanation: An error reading a segment block causes the message to be issued and the job is terminated.

ERROR USING GET ON ICALL MACRO. JOB TERMINATED

Explanation: DCRS could not get the next record from the ICALL macro.

**I/O ERROR OCCURRED IN READING THE
DIRECTORY. JOB TERMINATED**

Explanation: DCRS could not read the PDS's directory. Scanning terminates since the member names are unknown.

DREF Messages

**J, K, L, M, N, O, OR S NOT FIRST CHARACTER OF
INPUT RECORD.**

Explanation: DREF expects the first character of each entry in the input data set to be a code letter. This code

letter tells DREF what kind of data the entry represents and how it should be handled. The valid codes are: A, B, C, J, K, L, M, O, S, and X.

References

DFSORT Application Programming Guide

Multiple Assembly/Compilation Program

The Multiple Assembly/Compilation Program (MASM) allows you to assemble or compile your programs, which are in partitioned data sets (PDSs), with minimum usage of JCL. MASM provides several features and options to customize your environment:

- Choose between the High-Level Assembler (HLASM) or an IBM OS/390 C/C++ compiler.

Note: See the *TPF Migration Guide: Program Update Tapes* and *OS/390 C/C++ User's Guide* for more information about C and C++ compilers.

- Through control cards to MASM, any subset of the source library's segments can be selected for assembly or compilation.
- You can specify where you want listings sent. All listings, or only those that contain errors, can be sent to a SYSOUT device. In the later case, the listings with errors are sent to the SYSOUT device while the error-free listings are archived.
- Cross-reference listings can be generated with the object listings, or suppressed.
- E-type programs, offline programs, and CP segments can be assembled.
- E-type programs can be compiled.
- The level of C or C++ compiler error reporting can be changed.
- Additional system- or user-include libraries can be passed to a C or C++ compiler.
- MASM will set the return code to 4 if any program finds an error during assembly or compilation.

Note: To understand the following discussion, you need to understand MVS JCL.

Input

The following section describes the input to the MASM program.

Files

Input to MASM is the partitioned data set containing the programs to be assembled or compiled.

JCL Control Cards

MASM requires the following data sets when executed:

PDS	DD card for the source libraries.
MSGFILE	SYSOUT data set containing diagnostic messages from the program and the Assembly or Compilation Error Report.
DRIVEIN	Control card input files that select the program segments from the source library (PDS).

If you request the listings-to-tape option, MASM needs 2 more data sets defined:

LIST	DD statement for a SYSOUT device for listing of assemblies or compilations with errors.
LISTAPE	DD statement specifying a tape or disk to hold the error-free listings (see Figure 29 on page 394).

The data sets described so far are those used by MASM directly. All JCL cards normally required by either High-Level Assembler (HLASM) or a C or C++ compiler are still required when running MASM, though MASM, in a few cases, places extra requirements on them:

- The effects of MASM on the assembler:
 - When MASM calls the assembler, the DECK parameter is used. This places the object code into the data set specified by the SYSPUNCH data definition. However, MASM disables the SYSLIN data set by passing the NOOBJECT parameter to the assembler.
 - MASM sets the assembler LINECOUNT to 55. See *High Level Assembler /MVS & VM & VSE Programmer's Guide* for detailed information about this and other assembler parameters.
 - For a non-E-type program assembly, the SYSIN DD statement should point to the same library as the PDS DD statement (see Figure 28 on page 394).
 - If you want to perform an E-type assembly, specify PARM=RT on the EXEC card, specify a temporary data set in the SYSIN DD statement, and concatenate the source library with the SYSLIB data set (see Figure 30 on page 395). RENT should also be specified with RT to check for reentrancy.
 - As in any assembly, you must ensure that SYSLIB refers to the appropriate macro, and if necessary, the appropriate source libraries.
 - When the LIST and LISTAPE data sets are used, the assembler listings are converted from fixed block machine (FBM) length 121 to fixed block ANSI (FBA) length 133.
- The effects of MASM on the compiler:
 - When MASM calls a C or C++ compiler, the object deck is placed into the SYSLIN data set. The SYSPUNCH data set is not defined by the C or C++ compiler.
 - The SYSLIB data set must point to the TPF-supplied C language headers.

Attention: Do not add either the VM or MVS C language header data set to SYSLIB.

These data sets have been deliberately omitted from the SIP-generated C compilation procedures. The VM and MVS headers provide some functions that are incompatible with the TPF system environment. If these headers were added, the C program segment might compile without errors or warnings, but would not be usable in a TPF system environment.
 - MASM sets the following C or C++ compiler options that you cannot change: LIST, OFFSET, SOURCE, NOSTART, NOANSIALIAS, and AGGRC(OVERLAY). You can change the FLAG, LSEARCH, SEARCH, RENT, OPTIMIZE, and LONGNAME options. All other parameters are allowed to default. See your compiler User's Guide for detailed information about these and other compiler options.
 - OPT(*n*) passes the OPTIMIZE option to the compiler exactly as you code it. Valid values for *n* are OPT(0), OPT(1), and OPT(2). The default is OPT(2). You can also use OPT and NOOPT.

Note: See the *OS/390 C/C++ User's Guide* to understand what effect these parameters have on the IBM OS/390 compiler that you are using. For example, on some compilers OPT(1) is the same as OPT(2), and OPT has the same effect as OPT (1).
 - Either the RENT or NORENT option can be specified for a C or C++ program.
 - For E-type compilations, specify PARM=RT on the EXEC card and specify a temporary data set in the SYSIN DD statement (see Figure 31 on page 396).

- When the LIST and LISTAPE data sets are used, the compiler listings are converted from variable block length 137 to fixed block ANSI (FBA) length 133.

Execution Parameters

The following are the user options that are available as execution parameters:

TAPE	Error-free assembly listing are put to tape. Note: This requires you to include a LISTAPE and LIST DD statement and to change the SYSPRINT (or SYSCPRT for C) DD statement. See “Multiple Assembly/Compilation Print Program” on page 377 for the subsequent listing of the tape.
SHORT	See your appropriate assembler user’s guide. (Not valid for a C or C++ compiler.)
XREF	See your appropriate assembler user’s guide.
NOXREF	See your appropriate assembler user’s guide. NOXREF is the default value. If XREF is requested, it is expanded to XREF(FULL) by MASM.
RT	Indicates to MASM that real-time programs are being assembled or compiled. Note: Requires a temporary data set for the SYSIN data definition. For the assembler, the source libraries must be concatenated to the SYSLIB data definition. For a C or C++ compiler, the only data sets that can be concatenated are data sets containing C language headers.
I	MASM passes the program to the High Level Assembler (HLASM).
RENT	Passes the RENT option to the assembler or C or C++ compiler. This causes the assembler or C compiler to verify that a segment is reentrant. The RENT option is ignored if you specify the TARGET(TPF) option. Note: C++ programs are always compiled as reentrant regardless of whether you specify RENT or NORENT.
NORENT	Passes the NORENT option to the assembler or C or C++ compiler. The assembler or C or C++ compiler will not do reentrancy checking. The NORENT option is ignored if you specify the TARGET(TPF) option. Note: C++ programs are always compiled as reentrant regardless of whether you specify RENT or NORENT.
G	MASM passes the program to the IBM OS/390 C/C++ compiler and the program is compiled as C code.
GPP	MASM passes the program to the IBM OS/390 C/C++ Release 2 or Release 3 compiler and the program is compiled as C++ code.
DLL	Passes the DLL option to the compiler. This option is valid only for the IBM C/C++ for MVS/ESA Version 3 Release 2 and IBM OS/390 C/C++ compilers. This option is valid for C programs only; the parameter is ignored for C++ programs.

NODLL	Passes the NODLL option to the compiler. The parameter is ignored for C++ programs.
FL(x)	A C or C++ compiler FLAG option, where x can equal I, W, E, or S. Note: The value of x defaults to E for the C++ compiler.
LSE(yyyyy)	A C or C++ compiler LSEARCH option, where yyyyy can be any MVS data set containing user include files.
SE(zzzzz)	A C or C++ compiler SEARCH option, where zzzzz can be any MVS data set containing system include files.
OPT(n)	Passes the OPTIMIZE option to the compiler exactly as you code it. Valid values for n are OPT(0), OPT(1), and OPT(2). The default is OPT(2). You can also use OPT and NOOPT. Note: See the <i>OS/390 C/C++ User's Guide</i> to understand what effect these parameters have on the IBM OS/390 compiler that you are using. For example, on some compilers OPT(1) is the same as OPT(2), and OPT has the same effect as OPT (1).
LONGNAME	Passes the LONGNAME option to the compiler.
NOLONGNAME	Passes the NOLONGNAME option to the compiler.
TARGET(TPF)	Passes the TARGET(TPF) option to the C or C++ compiler. This parameter must be specified for TARGET(TPF) E-type programs and must not be specified for C load module E-type programs.

Control Card Options

The control card options provide the following facilities:

- Assemble or compile the entire directory (no control cards)
- Assemble or compile a list of programs
- Assemble or compile from a specified program to the end of the directory
- Assemble or compile from the beginning of the directory to and including a specified program
- Assemble or compile a range of programs in a directory
- Attach a suffix to all object module names

Option 1: Assemble or compile the entire directory

If no control cards are specified, all programs in the PDS directory will be assembled/compiled. The DRIVEIN DD card may appear as follows:

```
//DRIVEIN DD DUMMY
or:
//DRIVEIN DD *
/*
```

Option 2: Assemble or compile a list of programs

Format:

```
LIST=name
LIST=(name1,name2,name3,.....name9)
```

One or more programs for C and one program for C++ can be supplied in the LIST option. If there is only one program in the list, parentheses are not required. The program names are not required to be in any order; however, it is recommended that they appear in the order found in the directory (alphabetic).

A list can appear on several cards. The following example shows a list continued on more than one line:

```
LIST=(BMGL,BMPOX,CPTX,DMP0,  
      DMP1,SNXPD,  
      ZAPX,ZAPY,ZAPZ)
```

Note: A list must be enclosed in parentheses (if more than one name). If continued on the next card, the last name of the previous card must be followed by a comma. There are no column restrictions.

Option 3: Assemble or compile FROM option

Format:

FROM=name

This option allows one to assemble or compile all programs beginning with the specified FROM name through the end of the directory.

Option 4: Assemble or compile TO option

Format:

TO=name

This option allows one to assemble or compile all programs from the beginning of the directory to and including the specified TO name.

Option 5: Assemble or compile a range of programs

Format:

FROM = name1,TO = name2

This option will assemble or compile all programs in the directory from name1 through name2 inclusive.

Option 6: Attach a SUFFIX

Format:

SUFFIX = xx

This option allows you to specify a suffix that will be attached to the object module name. The suffix can contain as many as 6 alphanumeric characters. There is one requirement for this option. It must precede the option (1-5) for which it is to apply.

Note: You must consider the final length of the concatenated name, which cannot exceed 8 characters. If greater than 8, the program will not be assembled or compiled and a diagnostic will appear.

Options 1-5 are considered as individual tasks. One task can appear on one line. The suffix parameter can precede any or all tasks requested.

Examples of Control Card Options:

Example 1: LIST=BMP0

Assemble or compile the program called BMP0.

Example 2: SUFFIX=53,TO=DGR0

Assemble or compile all programs from the beginning of the directory to and including the program DGR0. Attach the suffix 53 to all object module names.

Example 3: SUFFIX=54

Assemble or compile the entire directory, attaching the suffix 54 to all object module names.

Example 4: LIST=(BMP0,CRAS,DGR0)

Assemble or compile the 3 programs specified in the list.

Example 5:

```
LIST=(BMP0,  
      CRAS,  
      DGR0)
```

An alternate way of showing Example 4.

Example 6:

```
TO=CRAS  
LIST=(DGR0,ASMM,ZAPX,ZAPY)
```

Task 1:

Assemble or compile from the beginning of the directory to and including CRAS.

Task 2:

Assemble or compile the programs specified in the list.

Example 7:

```
SUFFIX=44,LIST=(BMP1,BMP2)  
      FROM=ZAPA,  
      TO=ZAPF  
SUFFIX=55,LIST=CRAS
```

Task 1:

Assemble or compile programs BMP1 and BMP2, attaching the suffix 44 to the object modules.

Task 2:

Assemble or compile all programs between ZAPA AND ZAPF inclusive (no suffix requested).

Task 3:

Assemble or compile the program CRAS attaching the suffix 55.

User Considerations

Following are some helpful hints and considerations for using the Multiple Assembly/Compilation Program.

- Use the TIME parameter if your system has a maximum time limit for a job. In estimating assembly or compilation time, consideration should be given to the total number of assemblies or compilations requested.
- Under MVS, there is the possibility that the SYSOUT data set may run out of space if many assemblies or compilations are done in one run. Either provide more space for the SYSOUT data set (SYSPRINT DD card for assembler, or SYSCPRT DD card for C), or go directly to the printer.
- Compile only one C++ program at a time with the LIST=control card because the compiler creates names for static initialization routines based on the job name and time stamp of the job. If more than one program has a static initialization routine and those programs are linked together at a later time, the link-edit will fail because the names for all static initialization routines will be the same.
- Check CAREFULLY the macro libraries assigned to the SYSLIB DD. This is not new to anyone running assembly jobs. However, when many assemblies are being run, the entire run could be wasted if the wrong macro library is used.
- When using the TAPE parameter be sure to provide a tape initialized in the same manner as the LISTAPE data definition specifies (for example, standard label, device type, and others).
- If desired with the TAPE parameter, the LISTAPE data definition can specify a DASD. If so, carefully calculate the amount of space needed based on number, size, and whether XREF is requested. The DCB definition is preset at RECFM=FBA and LRECL=133 with the BLKSIZE user variable based on needs (a multiple of LRECL).

Output

Output from the MASM program is in the form of listings or files.

Listings

The listings produced by the Assembler or Compiler invoked will be produced either on tape or on the printer, as specified by JCL execute parameter.

Files

For each program assemble, an object module will be written as a member of the SYSPUNCH data set.

For each program compiled, an object module will be written as a member of the SYSLIN data set.

Sample JCL

The following sample JCL is provided as examples of how to code MASM JCL statements.

Printout Directory to Listings

The sample JCL that follows is designed to assemble programs MEMBER1, MEMBER5, and all programs between MBR72 and MBR89 from cataloged library TESTCASE.LIBRARY, and the object modules will be output into a cataloged library called TESTCASE.LIB2. A cross-reference is produced.


```

//STEPSA EXEC PGM=MASM,REGION=200K,TIME=20,PARM=I
//STEPLIB DD DISP=SHR,DSN=ACP.LINK.RELv.vv.SSname
//SYSLIB DD DISP=SHR,DSN=SYS1.MACLIB
//SYSUT1 DD DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT2 DD DSN=&&SYSUT2,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT3 DD DSN=&&SYSUT3,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD DISP=OLD,DSN=TESTCASE.LIB2
//SYSIN DD DISP=SHR,DSN=TESTCASE.LIBRARY
//PDS DD DISP=SHR,DSN=TESTCASE.LIBRARY
//MSGFILE DD SYSOUT=A,DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//DRIVEIN DD *
LIST=(MEMBER1,MEMBER5)
FROM=MBR72,TO=MBR89
/*

```

Figure 28. Printout Directory to Listings JCL

In Figure 28, note the following:

- The STEPLIB data definition refers to the release link library.
- The vv refers to the actual release number and ssname refers to the subsystem name.
- Both PDS and SYSIN point to the source PDS.

Assembly Listings to Tape

The sample JCL that follows will assemble all programs in cataloged library TESTCASE.LIBRARY and put the object modules into cataloged library TESTCASE.LIB2. Printouts are put to tape if the assemblies are error-free. Consequently the SYSPRINT DD statement specifies a temporary data set on DASD and a DD statement is included for output tape LISTAPE. PARM=TAPE is specified on the EXEC card. Data definition LIST defines the output device to which programs containing assembly errors will be routed.

```

//STEPSA EXEC PGM=MASM,REGION=200K,TIME=20,PARM=(I,TAPE)
//STEPLIB DD DSN=ACP.LINK.RELv.vv.SSname,DISP=SHR
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSPRINT DD DSN=&&PA,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=3630),
// UNIT=SYSDA,SPACE=(CYL,(2,1))
//LISTAPE DD DSN=PRINT,UNIT=TAPE,DISP=(NEW,KEEP),
// VOL=SER=123456,DCB=(BLKSIZE=23940,LRECL=133,RECFM=FBA),
// LABEL=(,SL)
//LIST DD SYSOUT=A,DCB=(BLKSIZE=7980,LRECL=133,RECFM=FBA)
//SYSPUNCH DD DSN=TESTCASE.LIB2,DISP=OLD
//PDS DD DSN=TESTCASE.LIBRARY,DISP=SHR
//MSGFILE DD SYSOUT=A,DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//SYSIN DD DSN=TESTCASE.LIBRARY,DISP=SHR
//DRIVEIN DD DUMMY
/*

```

Figure 29. Assembly Listings to Tape JCL

In Figure 29, note the following:

- The STEPLIB data definition refers to the release link library.
- The vv refers to the actual release number and ssname refers to the subsystem name.

- If you expect a considerable amount of assembly output, you can specify more than one tape in the LISTAPE data definition (that is, VOL=SER=(123456,789012)).
- To put assembly listing output to DASD, specify DASD in the LISTAPE data definition.

Note: Make sure you allocate enough DASD space based on the size and number of programs being assembled.

- Successive executions of MASM can be accomplished by modifying the LISTAPE data definition (that is, DISP=(MOD,KEEP)).
- Both PDS and SYSIN point to the source PDS.

E-Type Assemblies to Tape

The control cards in the following example specify that only members MBR1, MBR2, MBR30, and MBR3 of library TESTCASE.LIBRARY are to be assembled. It attaches suffix 01 to the object code member names placed into TESTCASE.LIB2.

Because this is an E-type assembly, PARM=RT,RENT was specified on the EXEC card. PARM=I was used to obtain the High Level Assembler (HLASM). The SYSIN DD statement points to a temporary data set on DASD and the source library is concatenated into the SYSLIB data set. Care must be taken in cases like this to ensure that the first data set in the concatenation has the largest blocksize. Because the High Level Assembler (HLASM) is used, a region of at least 200KB must be used for the assembly.

```
//STEPS EXEC PGM=MASM,REGION=220K,TIME=20,PARM='I,RT,RENT,TAPE'
//STEPLIB DD DSN=ACP.LINK,RELvv.SSname,DISP=SHR
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
// DD DSN=TESTCASE.LIBRARY,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSPRINT DD DSN=*&PA,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=3630),
// UNIT=SYSDA,SPACE=(CYL,(2,1))
//LISTAPE DD DSN=PRINT,UNIT=TAPE,VOL=SER=123456,LABEL=(,SL),
// DISP=(NEW,KEEP),DCB=(BLKSIZE=23940,LRECL=133,RECFM=FBA)
//LIST DD SYSOUT=A,DCB=(BLKSIZE=7980,LRECL=133,RECFM=FBA)
//SYSPUNCH DD DSN=TESTCASE.LIB2,DISP=OLD
//SYSIN DD SPACE=(CYL,5),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3600),
// UNIT=SYSDA
//PDS DD DSN=TESTCASE.LIBRARY,DISP=SHR
//MSGFILE DD SYSOUT=A,DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//DRIVEIN DD *
          SUFFIX=01
          LIST=(MBR1,MBR2,MBR30,MBR3)
/*
```

Figure 30. E-Type Assemblies to Tape JCL

In Figure 30, note the following:

- The STEPLIB data definition refers to the release link library.
- The vv refers to the actual release number and ssname refers to the subsystem name.
- If error-free assemblies are to be put to tape, LIST and LISTAPE DD statements should be included and the SYSPRINT DD statement changed to a temporary data set as in Figure 29 on page 394.
- PARM=TAPE must also be specified on the EXEC card.

E-Type Compilations Sent to the Printer

The following example shows MASM calling the AD/Cycle C/370 compiler. The PARM parameter is coded with a D to call the AD/Cycle C/370 compiler, with an RT to designate E-type compilations, and with NOXREF to suppress the compiler cross-reference. The compiler requires a region of at least 4MB and several work data sets (SYSUT1–SYSUT9). SYSUT10 is coded as a dummy because the compiler option that uses it is suppressed by MASM. Although this job generates E-type object code, the source code libraries should not be concatenated to the SYSLIB data set. The compiler expects that only header libraries will be concatenated to SYSLIB.

```
//COMPILE EXEC PGM=MASM,REGION=4M,
//  PARM=(D,RT,NOXREF,TARGET(TPF))
//STEPLIB DD DSN=ACP.LINK.RELvV.SSname,DISP=SHR
//        DD DSN=LE.V1R3M0.SCEERUN,DISP=SHR
//        DD DSN=PLI.V2R3M0.SIBMLINK,DISP=SHR
//        DD DSN=EDC.V1R2M0.SEDCDCMP,DISP=SHR
//SYMSGSGS DD DSN=EDC.V1R2M0.SEDCDMSG(EDCMSGE),DISP=SHR
//SYSIN DD DSN=&&SYSIN,UNIT=SYSDA,
//        SPACE=(CYL,(3,3),RLSE),DISP=(NEW,DELETE,DELETE),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=3600)
//SYSLIB DD DSN=ACP.CHDR.RELvV.SSname,DISP=SHR
//SYSLIN DD DSN=ACP.OBJ.RELvV.SSname,DISP=OLD
//SYSPRINT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,SPACE=(32000,(30,30)),
//        DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD DUMMY
//*
//MSGFILE DD SYSOUT=A,
//        DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//PDS DD DSN=ACP.CSRCE.RT.RELvV.SSname,DISP=SHR
//DRIVEIN DD *
//        LIST=(C024vV,C025vV,C026vV,C027vV,C028vV)
//*
```

Figure 31. E-Type Compilations Sent to the Printer JCL

In Figure 31, note the following:

- The STEPLIB data definition refers to the release link library.
- The vv refers to the actual release number and ssname refers to the subsystem name.
- SYSIN must be a temporary data set.
- If error-free compiles are to be put to tape or disk:
 - Specify PARM=TAPE on the EXEC card.
 - LIST and LISTAPE DD statements should be included.

- SYSPRT DD statement should be changed to a temporary data set as follows:

```
//SYSPRT DD DSN=&&PA,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=882),
//          UNIT=SYSDA,SPACE=(CYL,(2,1)),SYSOUT=
```

Note: The SYSOUT parameter is required only when overriding a SYSPRT DD statement containing a SYSOUT parameter in a cataloged procedure. It is set to blank intentionally and must be coded as shown.

E-Type Compilations Sent to the Printer

The following example shows MASM calling the AD/Cycle C/370 compiler to compile C load module E-type programs with optimization level 2. The PARM parameter is coded with a D to call the AD/Cycle C/370 compiler, with an RT to designate E-type compilations, and with NOXREF to suppress the compiler cross-reference. The compiler requires a region of at least 4MB and several work data sets (SYSUT1–SYSUT9).

```
//COMPILE EXEC PGM=MASM,REGION=4M,
// PARM=(D,RT,NOXREF,OPT(2))
//STEPLIB DD DSN=ACP.LINK.RELv. SSname,DISP=SHR
//          DD DSN=LE.V1R3M0.SCEERUN,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMLINK,DISP=SHR
//          DD DSN=EDC.V1R2M0.SEDCDCMP,DISP=SHR
//SYMSGSGS DD DSN=EDC.V1R2M0.SEDCDMSG(EDCMSGE),DISP=SHR
//SYSIN DD DSN=&&SYSIN,UNIT=SYSDA,
//          SPACE=(CYL,(3,3),RLSE),DISP=(NEW,DELETE,DELETE),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3600)
//SYSLIB DD DSN=ACP.CHDR.RELv. SSname,DISP=SHR
//SYSLIN DD DSN=ACP.OBJ.RELv. SSname,DISP=OLD
//SYSPRINT DD SYSOUT=*
//SYSPRT DD SYSOUT=*
//SYSUT1 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,SPACE=(32000,(30,30)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD DUMMY
//*
//MSGFILE DD SYSOUT=A,
//          DCB=(RECFM=FBM,LRECL=80,BLKSIZE=80)
//PDS DD DSN=ACP.CSRCE.RT.RELv. SSname,DISP=SHR
//DRIVEIN DD *
//          LIST=(CSCNFvv,CTZDIFvv)
//          /*
//          //
```

Figure 32. E-Type Compilations Sent to the Printer JCL

In Figure 32, note the following:

- The STEPLIB data definition refers to the release link library.
- The vv refers to the actual release number and ssname refers to the subsystem name.

Error Messages

The following MASM messages can help determine MASM problems.

INVALID PARM PARAMETER

Explanation: An unrecognized argument was found on the JCL EXEC PARM parameter. See "Execution Parameters" on page 389 for a list of valid arguments.

THE COMBINED C OPTIONS EXCEED MAXIMUM

Explanation: The number of characters coded for the FL, LSE, SE arguments on the JCL EXEC PARM will cause the PARM buffer to overflow. The combined lengths of these arguments (including the commas and parentheses) must be less than 60.

INVALID KEYWORD ON CONTROL CARD

Explanation: A keyword on a control card is not one of the following:

LIST, FROM, TO or SUFFIX

END OF CARD REACHED - VALUE OR KEYWORD INVALID

Explanation: Column 72 of the card has been reached while scanning an entry on a control card and a delimiter has not been found.

NO CORE FOR LIST - INCREASE REGION SIZE

Explanation: A LIST option has been found on a control card and a GETMAIN macro issued to obtain core to build this list. The requested core is not available. This can be remedied by increasing the region size.

MISSING RIGHT PAREN IN LIST - ASSUMED CONTINUE

Explanation: A list of members on a list control card has not been terminated by a right parenthesis. A right parenthesis is assumed.

VALUE OF KEYWORD MISSING

Explanation: A keyword - LIST, FROM, TO or SUFFIX has been found without any value indicated.

INVALID - FROM - OPTION, EXPECTING FOLLOWING - TO -

Explanation: A FROM keyword and value has been found without a corresponding TO option.

SUFFIX GREATER THAN 6 CHARACTERS

Explanation: A suffix has been supplied via a control card which exceeds 6 characters.

LENGTH OF FROM/TO VALUE GREATER THAN 8 CHARACTERS

Explanation: A value assigned to a FROM or TO keyword exceeds 8 characters.

MAXIMUM (n) NO. OF MEMBERS FOR LIST EXCEEDED

Explanation: n members are allowed in a list. The list supplied on a control card exceeds this number. To correct this problem the local variable &MAXMEM in ASSMDRIVE should be increased and the program reassembled.

SUFFIX + MEMBER NAME GREATER THAN 8 CHAR

Explanation: When the suffix has been added to the input library member name the resultant name exceeds 8 characters.

PROGRAM NOT FOUND

Explanation: The DRIVERIN data set has specified a program segment that was not a member of the source code data set.

Hardware Requirements

An operating system (hardware and software) capable of supporting the High Level Assembler (HLASM) and a C or C++ compiler with enough additional direct access file space to store the input and output partitioned data sets is required.

System Allocator

The system allocator (SALO) is an offline program that creates the system allocator (SAL) table and the program allocation table (PAT). To help with this section a few definitions are provided as follows:

dummy	Function switches can be coded on each card in the SALO input deck. If the function switch is off, the program is dummy allocated and, therefore, the program is not entered and no DASD is required. These entries reside in the SAL table only.
parent	The program in which a transfer vector is located. The parent can have multiple entry points (transfer vectors) in it.
spare	Programs allocated as spare serve as placeholders that can be defined for future application expansion. If an application is removed, the placeholders are needed to avoid a shift of the remaining applications.
transfer vector	The label name given to an entry point into a program.

SALO builds the SAL table and PAT from concatenated input decks as shown in Figure 33 on page 400. The input decks contain program names and characteristics; for example, where the programs are to reside, what addressing mode they operate in, and whether they have any privileges. The input decks also contain symbol definition cards that are used to resolve V-type address constants (V-cons). As many as 16 input decks can be concatenated. The IBM input deck, IBMPAL, is concatenated first. You can specify as many as 15 more user input decks to hold user allocation information.

The output tables from SALO (SAL table and PAT) contain the information from the input decks in the form required by the linkage editor (LEDT) and online programs.

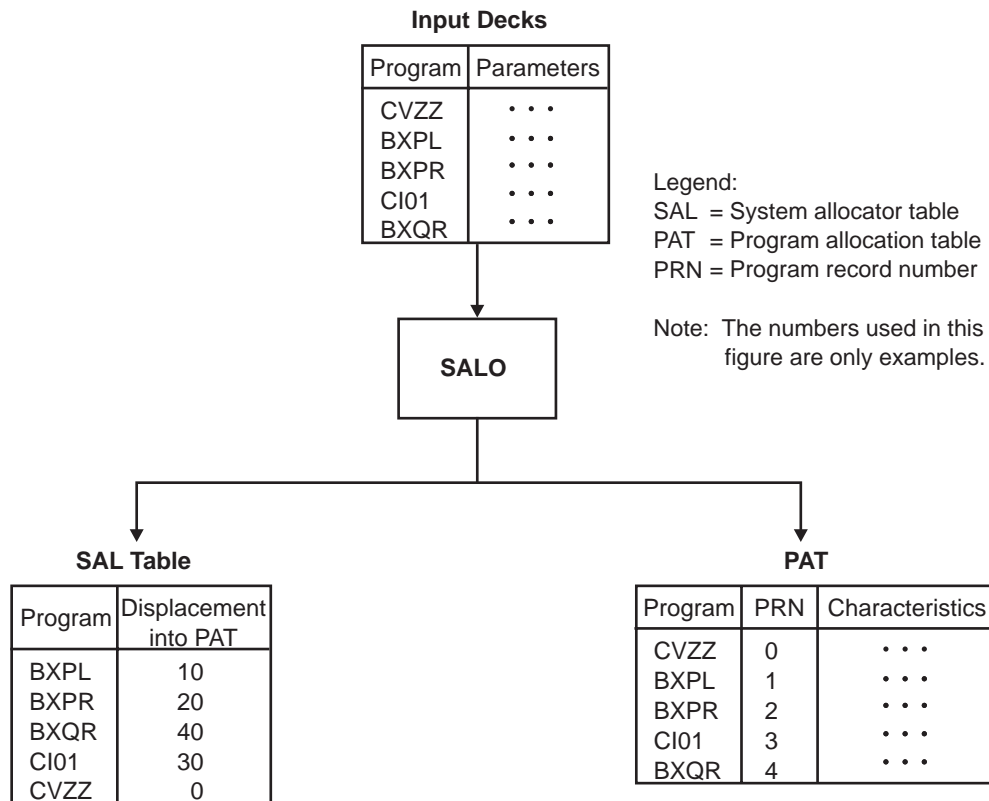


Figure 33. Operation of SALO

The SAL table:

- Resolves the offline external references for the TPF linkage editor. The SAL table holds the program's name and its displacement into the PAT (determined by the order in which programs are listed in the input deck).

The PAT:

- Keeps track of the file or main storage locations for all E-type programs
- Is used by Enter/Back to determine each program's characteristics
- Keeps track of which programs were activated by the E-type loader.

To summarize, SALO is an offline tool for creating the SAL and PAT tables.

Allocating Programs, Transfer Vectors, and Pools

There are significant changes to the system allocator (SALO) process in the TPF 4.1 system:

- The offline SALO is written in IBM C Language, and is compiled, link-edited and executed in one job.
- SKPAL and PAL are no longer supported. They are replaced by SALO input decks. SALO uses these decks to create the program allocation table (PAT) and the system allocator (SAL) table.

There can be up to 16 input decks. IBM provides one input deck, which is the IBMPAL input deck. You can concatenate up to 15 input decks of your own. Do not add allocator statements to the IBMPAL input deck because the order of the programs in the decks determines their ordinal number (also known as the program

record number (PRN)) in the 4KB fixed-file record area. However, you may need to modify the allocator statements to meet your needs. All programs are allocated as 4KB.

You do not need to put all of the required core resident (CR) programs at the front of the allocator decks because residency is specified on the allocator statement itself, rather than being determined by the position of the allocator statement.

You need to specify the names of the input decks and the data set that contains them on the new INDSN macro using these parameters:

SALDSN

The name of the input data set that contains all of your input decks (as members) for the system allocator.

SALMEM

A sublist of up to 15 deck (member) names. These decks, along with system required decks, are used as input to the system allocator.

Creating the Input Deck

The following sections describe how to code user input decks. The decks consist of SALO input statements that contain program allocation information, symbol definitions, default mode settings, or comments. All statements, except comments, must be coded in uppercase.

All program, transfer vector, and symbol definition statements are placed in the SAL table. All program and spare statements also appear in the PAT unless function switches or subsystem switches coded for them are false.

Programs can be allocated either normally or as dummy. Normally allocated programs appear in both the SAL table and the PAT. Dummy allocated programs only appear in the SAL table and indicate to the linkage editor (LEDT) that the name of the dummy allocated program is valid and should not be flagged as an unresolved external.

A program is allocated as dummy if the function switch name coded on the FUNC= parameter of a program allocation statement resolves to 0 (false) **or** the subsystem name coded on the SS= parameter of a program allocation statement does not match the subsystem name coded on the SS= parameter passed to SALO.

Function switch names are resolved by the SIP created C header file c\$idfunc. c\$idfunc equates a 1 or a 0 (true or false) with each valid function switch name.

Comments

You can intersperse comments with input statements but they must contain an asterisk (*) in column 1. You can also put comments on the same line as other statements but at least one space must separate the comment from the statement.

Specifying the Addressing Mode

When you want a block of programs to run in a particular addressing mode (24-bit addressing mode or 31-bit addressing mode) without specifying this on each individual input statement, you can use the MODE statement. This statement provides a default mode setting for all subsequent program statements in the input decks.

The format of the MODE statement follows:
where **24BIT** is the default.

MODE=24BIT|31BIT

Notes:

1. All programs written in the IBM C Language must be allocated in 31-bit addressing mode. This allows them to call heap storage functions, which return 31-bit addresses.
2. The FACE, SIGT, RIAT, and SNA tables are now above 16MB. All programs that manipulate these tables or reference the FC0TB, SI0GT, and DCTRIT data macros must be allocated as 31-bit addressing mode.

Allocating Programs

All program records are 4KB. The order in which the programs appear in the input decks determines their program record numbers (PRNs). Every program allocation statement is allocated as a 4KB block.

The format of the input statements for allocating file resident (FR) and main storage resident or core resident (CR) programs follows:
where:

```
prog,FR|CR [,CLASS=SHARED|COMMON|UNPROT|ISUNIQ|PRIVATE] [,MODE=24BIT|31BIT]
[,OPTIONS=(KEY0,MONTC,RESTRICT,CMB)[,SS=ALL|EXC|ssname]
[,FUNC=1|0|fsname]
[,PRELOAD ]
[, NODBUG ]
```

prog

The 4-character program name that is being allocated. The program name must be four characters long and the first character must be alphabetic.

FR|CR

Indicates whether the program is to be file resident (FR), main storage resident, or core resident (CR).

Notes:

1. FR replaces the MR used in previous releases.
2. Core resident (CR) programs do not have a demand counter.
3. Core resident programs must have CLASS values of SHARED or COMMON.

CLASS=

The class of program can be SHARED, COMMON, UNPROT, ISUNIQ, or PRIVATE. The default is **SHARED**.

SHARED

Indicates that all ECBs use the same copy of the program, but only those ECBs currently using the program can view it. In addition, all ECBs using the program can potentially access the program at different ECB virtual addresses. The program resides in page-protected memory and key-protected memory. The demand counter for noncore SHARED programs is located in the PAT entry of the program.

COMMON

Indicates that all ECBs use the same copy of the program and access the program at the same address. The program resides

in key-protected storage. The demand counter for a noncore COMMON program is located in the PAT entry of the program.

UNPROT

Indicates that this is a self-modifying I-stream shared program. All ECBs use the same copy of the program and access the program at the same address. The program resides in unprotected storage. The demand counter for an UNPROT program is in the 8-byte program header.

ISUNIQ

Indicates that this is a self-modifying I-stream unique program. All ECBs on the same I-stream use the same copy of the program and access the program at the same address. All copies of the program reside in unprotected storage. The demand counter for an ISUNIQ program is in the 8-byte program header.

Moreover, ISO-C programs cannot be allocated as private or as IS-unique.

PRIVATE

Indicates that this is a private program to be loaded to a unique unprotected working storage block each time it is entered. PRIVATE programs do not have a demand counter.

ISO-C programs cannot be allocated as private or as IS-unique.

Table 7 summarizes the use of the CLASS attribute for file-resident programs.

Table 7. Summary of CLASS Attribute for File Resident Programs

	I-Stream Shared	I-Stream Unique	ECB Unique	Same Address	Protected	Unprotected
SHARED	X				X	
COMMON	X			X	X	
UNPROT	X			X		X
ISUNIQ		X		X		X
PRIVATE			X			X

MODE=24BIT|31BIT

The addressing mode in which the program is entered. This parameter overrides any preceding MODE statement for this program. All programs written in IBM C Language must be allocated in 31-bit mode.

OPTIONS=

One or more of the following macro authorization options (separated by commas):

KEY0

Specifies that the E-type program can issue a macro that lets the program store into protected storage (change the protection key to 0).

MONTC

Specifies that the E-type program can issue a MONTC macro that lets the program store into protected storage and execute privileged instructions until the program issues an LMONTC macro.

RESTRICT

Specifies that the program can issue macros that are restricted for other reasons.

CMB

Specifies that the program can issue a macro to get common blocks.

SS=

A valid subsystem name as declared in the SIP Stage I SSDEF macro:

ALL Indicates that the program should be allocated in all subsystems.

EXC Indicates that the program should be allocated in all subsystems except the BSS.

ssname

A valid subsystem name as declared in the SIP Stage I SSDEF macro.

The default is **ALL**.

FUNC=

A function switch:

1 Specifies that the program allocated by this statement is placed in both the system allocator (SAL) table and the program allocation table (PAT).

0 Specifies that this program is a dummy and is put only in the system allocator (SAL) table.

fsname

A valid switch name taken from the SIP-created IBM C language header file *c\$idfunc.h*. See "Adding Your Own Function Switches" on page 406 for more information.

The default is **1**.

PRELOAD

Guarantees a core resident (CR) program will be loaded before 1052 state. The number of PRELOAD programs should be minimized for performance reasons.

NODBUG

Specifies that the program is not available to be debugged using the TPF Assembler Debugger for VisualAge Client. This parameter is available only for programs that are allocated as 31-bit core resident or file resident with a class of SHARED or COMMON.

Allocating Transfer Vectors

A *transfer vector* is the label name given to an entry point into a program. The program in which the entry point is located is called the parent. The parent can have multiple entry points (transfer vectors) within it. An allocation statement for the parent program must precede all its transfer vector's allocation statements. A transfer vector has the same allocation attributes as its parent. Each transfer vector uses 4KB of DASD.

Note: Transfer vectors are not supported for ISO-C dynamic load modules (DLMs). DLMs can have only one entry point.

The format of the input statements for allocating transfer vectors follows:
where:

<i>prog</i> , TV , <i>parent</i> , <i>tv</i> #

prog

The 4-character transfer vector name that is being allocated. The first character must be alphabetic.

TV

Indicates that this is a transfer vector statement.

parent

The 4-character program name this transfer vector is located in. The parent cannot be a transfer vector.

tv#

The transfer vector number for *prog*, which must be a decimal number in the range 0 to 1023.

Allocating Spare Program Slots

Spare statements interspersed throughout the program allocation table (PAT) hold program record numbers (PRNs) and 4KB blocks of DASD so that programs can be placed in these positions at a later time. This allows you to add and delete programs without changing existing PRNs. Any programs you add without using spares must be added after the last entry in the last input deck. If programs are added in the middle, a full load must be done.

The format of the input statements for allocating spare program slots follows:
where:

SPARE

SPARE

Indicates that this is a spare statement.

Defining Pools

The format of the input statements for defining pools follows:
where:

<i>poolsym</i> , TYPE=POOL,VALUE=svc

poolsym

The 3-character GFS ID to be expanded. The first two characters are the record ID and the third character is the record size attribute (L for large records or S for small records).

svc

A 3-byte hexadecimal SVC interrupt code. Valid codes include:

0ABC <i>nn</i>	SLT (small, long-term)
0ABE <i>nn</i>	SST (small, short-term)
0AC0 <i>nn</i>	SDP (small, duplicated)
0AC2 <i>nn</i>	LLT (large, long-term)
0AC4 <i>nn</i>	LST (large, short-term)
0AC6 <i>nn</i>	LDP (large, duplicated)

where *nn* stands for:

00	ratio dispensing option
04	DASD Device A
08	DASD Device B

0C DASD Device C

10 DASD Device D

Adding Your Own Function Switches

Function switches determine whether a program is allocated normally or as a dummy program.

To Add Your Own Function Switches

1. Update the SIP SKFUNC skeleton. Add your new switch in four places:
2. In the list of switches in the prologue section (comments) of the macro, along with a description of the switch, for example:

```
. *   &NEWFUNC           NEW FUNCTION SWITCH
```

3. In the section where global variables are defined, for example:

```
GBLB &NEWFUNC
```

4. In the section containing SETB instructions, for example:

```
&NEWFUNC     SETB     (&SBaaaa AND &SBbbbb)
```

(where &SBaaaa and &SBbbbb are defined function switches).

5. In the final section that punches out the IBM C Language header file, before the EOF statement, for example:

```
PUNCH '               "NEWFUNC",&NEWFUNC, '
```

1. Code your new function switch name on the FUNC parameter of the program allocation statement.
2. Code EXPRS=S on the SIP GENSIP macro to generate the IBM C Language header file c\$idfunc.h.
3. Rerun SIP Stage I.
4. Execute the JCL generated by step 3.

When adding a new function switch, avoid using names beginning with &SB. This naming convention is reserved for IBM use.

Running the System Allocator (SALO)

Before you can run the system allocator (SALO), you must have:

- The IBMPAL input deck
- Any input decks that you want concatenated (and the names of these decks coded on the INDSN macro)
- Any of your own authorization bits in the IBM C Language header file c\$idsa1o.h.
- Any of your own function switches in the SIP SKFUNC skeleton.

There are two different ways you can run SALO, depending on whether you run SIP Stage I, or both SIP Stages I and II.

To Run Only SIP Stage I

1. Code EXPRS=S on the SIP GENSIP macro.
2. Assemble the Stage I deck.
3. Run the JCL that was produced from SIP Stage I.

To Run Both SIP Stages I and II

1. SALO is compiled automatically, link-edited, and executed during Stage II.

When SALO runs successfully, it produces the system allocator (SAL) table and the program allocator table (PAT), which are written to disk as partitioned data sets.

Creating the Program and System Allocator Tables

The SAL1A2E program is used to create the program allocator table (PAT) and the system allocator (SAL) table. Figure 34 shows part of the JCL created from SIP Stage I that is used to run the SAL1A2E program. You may want to change some of the statements to suit your purpose.

```
//SAL1A2E EXEC EDCCLG,INFILE='ACP.CSRCE.OL.REL40(SAL040)',  
//          GPARM='PROG=5500 SS=BSS ECHO=YES SAL=YES CC=NO'  
//COMPILE.SYSLIB DD DSN=&VSCCHD&CVER&EDCHDRS,DISP=SHR  
//              DD DSN=ACP.CHDR.REL40,DISP=SHR  
//              DD DSN=ACP.SYMACRO.REL40.BSS,DISP=SHR  
//GO.SYSUDUMP DD SYSOUT=A  
//GO.INFILE DD DSN=ACP.SALIN.REL40(IBMPAL),DISP=SHR  
//              DD DSN=ACP.SALIN.REL40(USRTPF),DISP=SHR  
//GO.SALOUT DD DSN=ACP.SALTBL.REL40.BSS(TABLE40),DISP=SHR  
//GO.PATOUT DD DSN=ACP.SYSRCE.RT.REL40.BSS(IPAT40),DISP=SHR  
//GO.LSTFILE DD SYSOUT=A  
//GO.SALRPT DD SYSOUT=A
```

Figure 34. Sample JCL Created by Running SIP Stage 1

GPARM

The GPARM statement identifies the general parameters that are used by SAL1A2E EXEC.

PROG

The PROG parameter identifies the number of programs you want to allocate.

SS

The SS parameter determines the name of the subsystem.

ECHO

The ECHO parameter determines how the error listing is created. If the value supplied is YES, all input cards and errors are listed. If the value supplied is NO, only errors are listed.

SAL

The SAL parameter determines if a SAL report is created. The SAL report shows the characteristics of all tapes, pools, and programs in the SAL table. The value supplied can be YES or NO.

CC

The CC parameter determines if the output report files have carriage control characters. The value supplied can be YES or NO.

INFILE

The INFILE parameter identifies data sets that contain lists of programs that you want to allocate.

SALOUT

The SALOUT statement determines where the SAL table is put.

PATOUT

The PATOUT statement determines where the IPAT source is put. The source must be assembled to create the actual PAT.

LSTFILE

The LSTFILE statement determines where the error listing is put.

SALRPT

The SALRPT statement determines where the SAL report is put.

Variable Cross-Reference Listing

The variable cross-reference listing program (VCRS) scans any partitioned data set containing programs written in assembler or macro assembler language. It also prints a cross reference listing of all the global variable symbols used in the PDS, cross-referencing them to the members in which they are used. By use of control cards, VCRS can be made to scan a selected list of members, certain variable types, or both.

The variables found can either be sorted alphabetically or by global type (A, B, or C), or alphabetically in SIP (System Initialization Process) order. The PDS scanned can be concatenated data sets. Using SYSLIB data definitions, you can search for segments that are called by other members of the PDS being searched and that use the assembler COPY statement. VCRS ignores PDS members that are written in PL/I language (for example, data reduction and pool generation).

VCRS identifies, by name, a ***copied*** member that uses global symbols. The copied member is referred to by the NAME(COPY) statement where:

NAME The prime PDS member being searched.

COPY A PDS member using a global symbol that was copied (using the assembler COPY statement) by the prime PDS member.

VCRS also cross-references variable symbols that are used in macro language SET statements.

JCL Control Cards

Input to the program is the partitioned data set to be scanned. Sort modules from SYS1.LINKLIB and SYS1.SORTLIB are also required. If the code being scanned contains any COPY statements, the PDS containing the members to be copied must also be included as the SYSLIB data set.

Figure 35 on page 410 shows the JCL required to run the VCRS program.

```

//VCROSS      EXEC PGM=VCRSvv,REGION=256K,TIME=20
//STEPLIB     DD DSN=ACP.LINK.RELv.vssid,DISP=SHR
//MSG         DD SYSOUT=A
//SYSPRINT    DD SYSOUT=A
//SYSOUT      DD SYSOUT=A
//SORTLIB     DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTIN      DD UNIT=SYSDA,SPACE=(TRK,(5,10)),DCB=BLKSIZE=5600
//SORTOUT     DD UNIT=SYSDA,SPACE=(TRK,(5,10)),
//            DCB=(RECFM=FB,LRECL=28,BLKSIZE=5600)
//SORTWK01    DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//SORTWK02    DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//SORTWK03    DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//PDS         DD DISP=SHR,DSN=ACP.SYMACRO.RELv.vssid
//            DD DISP=SHR,DSN=ACP.MACRO.RELv.v
//            DD DISP=SHR,DSN=ACP.SIPGEN.RELv.v
//            DD DISP=SHR,DSN=ACP.SYSRCE.RELv.vssid
//            DD DISP=SHR,DSN=ACP.SRCE.CP.RELv.v      (note - BSS only)
//            DD DISP=SHR,DSN=ACP.SRCE.RT1.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.RT2.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.RT3.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.OL.RELv.v
//SYSLIB      DD DISP=SHR,DSN=ACP.SYMACRO.RELv.vssid
//            DD DISP=SHR,DSN=ACP.MACRO.RELv.v
//            DD DISP=SHR,DSN=ACP.SIPGEN.RELv.v
//            DD DISP=SHR,DSN=ACP.SYSRCE.RELv.vssid
//            DD DISP=SHR,DSN=ACP.SRCE.CP.RELv.v      (note - BSS only)
//            DD DISP=SHR,DSN=ACP.SRCE.RT1.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.RT2.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.RT3.RELv.v
//            DD DISP=SHR,DSN=ACP.SRCE.OL.RELv.v
//SYSIN       DD *
              LIST=(PROGRAM1,PROGRAM9)
              SORT=SIP
              GLOBAL=A
/*

```

Figure 35. JCL Required to Run VCRS

where:

vv The user's version of the TPF system

ssid The user's subsystem ID

BSS The basic subsystem.

Note: All library names are referred to as per the TPF System Initialization Process (see *TPF System Generation*).

The region or partition in which the job is run should be at least 82KB. A STEPLIB or JOBLIB card is needed if the VCRS program is in a user's library rather than in SYS1.LINKLIB. The rest of the JCL is described as follows:

MSG This DD statement is used for error messages to the user.

SYSPRINT This DD statement is used for messages from the MVS Sort/Merge program, which is invoked by VCRS.

SORTLIB This DD statement points to the data set in which the modules of the Sort/Merge program reside.

SORTIN This is a work data set through which VCRS passes records to the sort/merge program. A BLKSIZE value must be supplied.

SORTOUT This is a work data set in which VCRS receives sorted records from

the Sort/Merge program. You must supply DCB parameters with LRECL=28 plus a BLKSIZE value.

SORTWK01-3 These are DD statements for work space used by the Sort/Merge program.

PDS This is a DD statement for the data set to be scanned. In the example, it is a concatenated data set of all TPF source libraries produced from SIP Stage II. As an alternative, if the referenced programs in the SYSIN data definition resided in a single data set, only that data set need to be specified. Normal MVS restrictions apply to the maximum number of concatenated data sets.

SYSLIB This DD statement points to partitioned data sets in which VCRS looks for members named in COPY statements found in the primary input. Nested COPY statements are ignored, for example, code referred to by a COPY statement cannot have a COPY statement in itself. The SYSLIB DD statement is not needed if there are no COPY statements in the primary input. When searching TPF segments in the PDS data definition (when the PDS DD statement is included), the SYSLIB DD statement must include, at least, the library (ACP.SYMACRO.RELvv.ssid) that contains the SYGLB and SYSET segments. See *TPF System Generation* for more information.

SYSIN This is a DD statement for the control card data set. These control cards are described below, in Control Card Input.

The JCL for running this program can be simplified by using a cataloged or instream procedure. The sample JCL that follows shows use of an instream procedure:

```
//VCROSS  PROC TYPE=OL,REL=95
//VCROSS  EXEC PGM=VCRS,REGION=256K,TIME=20
//STEPLIB DD DSN=TPF.LINK.REL&REL
//MSG     DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSOUT  DD SYSOUT=A
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTIN  DD UNIT=SYSDA,SPACE=(TRK,(5,10)),DCB=BLKSIZE=5600
//SORTOUT DD UNIT=SYSDA,SPACE=(TRK,(5,10)),
//         DCB=(RECFM=FB,LRECL=28,BLKSIZE=5600)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,10,,CONTIG)
//PDS     DD DISP=SHR,DSN=TPF.SRCE.&TYPE..REL&REL
//SYSLIB  DD DISP=SHR,DSN=TPF.SYMACRO.REL&REL
//        DD DISP=SHR,DSN=TPF.SRCE.&TYPE..REL&REL
//SYSIN   DD DUMMY
//        PEND
//X       EXEC VCROSS
```

Control Card Input

The control card options provide the following facilities:

1. Scan the entire PDS for every type of global variable.
2. Scan a list of members for every type of global variable.
3. Scan the entire PDS for 1 to 3 types of variables.
4. Scan a list of members for 1 to 3 types of variables.
5. Sort and print the globals found in the order specified.

Scan the Entire PDS for Every Type of Variable

If no control cards are specified, all members in the PDS directory will be scanned for every variable type with default options. The SYSIN DD card can appear as one of the following:

```
//SYSIN      DD    DUMMY
```

or

```
//SYSIN      DD    *  
/*
```

Scan a List of Members

Format:

```
LIST=name  
LIST=(name1,name2,name3,...name(n))
```

One or more members can be supplied in the LIST option. If there is only one member in the list, parentheses are not required. The names are not required to be in any order.

A list can appear on several lines. The following example shows a list continued on more than one line:

Format:

```
LIST=(BMGL,BMP0,CRTBC,  
CRSM, CLWA,  
CZXP,  
CYYA)
```

Note: A list must be enclosed in parentheses (if more than one name is supplied). If continued on the next line, the last name of the previous line must be followed by a comma. Only columns 1–71 can be used.

Subset of Variable Types Option

Format:

```
GLOBAL=type  
GLOBAL=(type1,type2)  
GLOBAL=(type1,type2,type3)  
GLOBAL=(type)
```

where:

type = A, B, or C

One or more global variable types can be supplied in the GLOBAL option. If there is only one global type in the list, parentheses can be omitted. The types do not need to be in any order. Only the types specified will be searched for.

Note: The list must be enclosed in parentheses (if more than one type is supplied). If continued on the next line, the last type of the previous line must be followed by a comma. Only columns 1–71 can be used.

Print the Globals Found in the Order Specified

Format:

`SORT=how`

where:

`how` = TYP, or GLB, or SIP

- If `SORT=TYP`

The global variables found will be printed in alphabetic order in global type (first A, then B, then C).

- If `SORT=GLB`

The global variables found will be printed in global name alphabetic order, regardless of global type.

- If `SORT=SIP`

Only the SIP global variable found (for example, `&X...` and `&S...`) will be printed and in SIP alphabetic order (disregarding the first 3 characters of the name).

Option Defaults

If you do not request options (see Scan The Entire PDS For Every Type Of Variable), the LIST option is not in effect (will scan entire PDS data definition, except PL/I members) (see Scan a List of Members), the GLOBAL option is set to `GLOBAL=(A,B,C)` (see Subset Of Variable Types Option), and the `SORT=GLB` option (see Print The Globals Found In the Order Specified) is in effect.

Examples of Control Card Options

Example 1: `LIST=SYSEQ`

List all global variables used in member SYSEQ.

Example 2: `LIST=SYCON,GLOBAL=A`

List all type-A global variables used in member SYCON.

Example 3: `LIST=(EB0EB,RTCEQ,BACKC)`

List all global variables used in EB0EB, RTCEQ, and BACKC.

Example 4:

```
LIST=(EB0EB,
      RTCEQ,
      BACKC)
```

An alternate way of showing Example 3.

Example 5: `GLOBAL=(A,C)`

List all type-A and type C global variables in the PDS.

Example 6: `GLOBAL=(A, C)`

An alternate way of showing Example 5.

Example 7: `LIST=(ST0TM,CINFC),GLOBAL=(A,B)`

List all type-A and B global variables occurring in members ST0TM and CINFC.

Example 8: `SORT=GLB`

List all of the variables referred in alphabetic order. Can be used with any of the other examples.

Procedure

The JCL and control cards should be made into a deck and run as a batch job under MVS.

Output

A variable cross-reference listing is produced that lists the global variable symbols used in the input data set and the members in which they are used. An error message listing may also be produced.

Error Messages

Errors consist of 2 types - control card errors, and errors which result from local variable symbols defined in the VCRS program being too small (such as &VBLS).

EQUAL SIGN DOESN'T FOLLOW KEYWORD/CARD REJECTED

Explanation: This message occurs when a keyword on the control card is not followed by an equal sign.

END OF CARD REACHED/VALUE OR KEYWORD INVALID

Explanation: This message is printed when column 72 of the control card is reached on a scan without finding a delimiter for the keyword or keyword value being scanned.

INVALID KEYWORD ON CNTRL CARD

Explanation: The keyword is not LIST, GLOBAL, or SORT.

GLOBAL VALUE NOT A,B,C

Explanation: The value of the GLOBAL = field on the control card is not A, B, or C.

INVALID SYNTAX ON CNTRL CARD

Explanation: The syntax on the control card does not correspond to the rules specified in either Subset of Variable Types Option or Print the Globals Found in the Order Specified or Option Defaults.

RIGHT "PARENDS" MISSING ON CNTRL CARD/ASSUMED

Explanation: A right parenthesis is missing on the control card.

INSUFFICIENT CORE/INCREASE REGION SIZE

Explanation: A GETMAIN for core to be used as a work area cannot be satisfied.

MAX (nnn) NO. OF MEMBERS FOR LIST EXCEEDED/INCREASE &LISTMEM AND REASSEMBLE

Explanation: The LIST option was used to specify a LIST of members to be cross-referenced. The program only caters for nnn members. Increasing the value of local variable &LISTMEM and reassembly of VCRS will correct this error.

MAX (nnn) NO. OF CONTINUATION CARDS EXCEEDED/INCREASE &CONTCRD AND REASSEMBLE

Explanation: The program allows for nnn continuation cards in a statement. A statement has more than this number of continuation cards when this message is issued. The problem can be corrected by increasing the value of local variable &CONTCRD, reassembly of VCRS and re-execution.

CONTINUATION CNTRL CARD MISSING

Explanation: Continuation control card expected but not found.

ERROR RETURN CODE FROM SORT

Explanation: On return from the Sort phase of the program, General Register 15 contains a nonzero value indicating that an error occurred in the sort.

**MAX (nnn) NO OF ALLOWED VARIABLES
EXCEEDED - INCREASE &VBLS AND REASSEMBLE**

Explanation: The number of global variables, nnn, allowed for in the program is less than the number occurring in the member being processed when the message is issued. This can be remedied by changing the value of local variable &VBLS, reassembly of VCRS and re-execution.

**NO GLOBAL VARIABLES USED IN MEMBERS
SEARCHED**

Explanation: The members searched (PDS data definition as modified by LIST option) did not contain any Global variable definitions.

**COPY REQUEST FOR MEMBER xxxxxxxx IN
MEMBER yyyyyyyy IS A NESTED
REQUEST/IGNORED**

Explanation: See JCL-SYSLIB data definition.

BUFFER NOT AVAILABLE FOR xxxxxxxx

Explanation: Where xxxxxxxx is either SYSLIB (SYSLIB data definition or PDS (PDS data definition).

An OS GETBUF macro instruction returned an error code for the specified data set.

SORT VALUE NOT SIP TYP GLB

Explanation: Sort parameter contained unknown keyword.

xxxxxxx MEMBER NOT FOUND

Explanation: Member xxxxxxxx, specified via LIST option, not found in PDS data definition.

**COPY REQUEST FOR MEMBER xxxxxxxx IN
MEMBER yyyyyyyy NOT FOUND, IGNORED**

Explanation: Copy member xxxxxxxx, requested by PDS member yyyyyyyy, not found in SYSLIB data set.

Hardware Requirements

An Operating System (hardware and software) capable of supporting the Sort/Merge program. Enough additional direct access space is required to store the input and SYSLIB data sets.

References

TPF System Generation

Appendix. JCL Load Deck Examples

This appendix contains examples of load decks for the auxiliary loader (TLDR) and E-type loader (OLDR) offline segments. The format of the load decks vary depending on the type of offline loader and the storage medium used.

TLDR JCL to Load Components to GDS

The example that follows shows the JCL required to load offline data to the general data set (GDS) using the auxiliary loader offline segment (TLDR).

Note: *nn* is the release identification and *ssid* is the subsystem ID.

```
//TLDRGDS JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A,
//          USER=mvuser,PASSWORD=password
//TLDR EXEC PGM=TPFLDRnn,REGION=7000K,PARM='TLDR,SYS=ACP,CLMSIZE=4000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SALTBD DD DSN=ACP.SALTBL.RELnn.ssid,DISP=SHR
//OBJDD DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//ALDRCPD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//TLDROUT DD UNIT=SYSDA,VOLUME=SER=dasd,
//          DSN=mvuser.OUTPUT,SPACE=(TRK,(40,20)),
//          DCB=(RECFM=F,BLKSIZE=4095,LRECL=4095),
//          DISP=(OLD,KEEP,KEEP)
//TAPEOUT DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
//TEMPLOAD DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
//LOADLIST DD SYSOUT=*
//LOADSUM DD DSN=&&LOADSUM,DISP=(NEW,PASS),UNIT=SYSDA,
//          LRECL=133,SPACE=(TRK,(10,10)),RECFM=FBA
//SYSUDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
/*
//PRTSUM EXEC PGM=IEBTPCH
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=&&LOADSUM,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSUT2 DD SYSOUT=*,LRECL=133
//SYSIN DD *
PRINT PREFORM=A
/*
```

TLDR JCL to Load Components to Tape

The example that follows shows the JCL required to load offline data to tape using the auxiliary loader offline segment (TLDR).

Note: *nn* is the release identification and *ssid* is the subsystem ID.

```
//TLDRVTAP JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A
//TLDR EXEC PGM=TPFLDRnn,REGION=7000K,PARM='TLDR,SYS=ACP,CLMSIZE=4000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SALTBD DD DSN=ACP.SALTBL.RELnn.ssid,DISP=SHR
//OBJDD DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//ALDRCPD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//TLDROUT DD UNIT=SYSDA,DSN=&&TLDROUT,DISP=(NEW,PASS)
//TAPEOUT DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
```

```

//TEMPLOAD DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
//LOADLIST DD SYSOUT=*
//LOADSUM DD DSN=&&LOADSUM,DISP=(NEW,PASS),UNIT=SYSDA,
//          LRECL=133,SPACE=(TRK,(10,10)),RECFM=FBA
//SYSUDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
/*
//MKTAPE EXEC PGM=IEBGENER,REGION=58K
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD DSN=TLDR.TAPE,LABEL=(,SL),UNIT=3480,
//          VOL=SER=vvvvvv,DISP=(,KEEP),
//          DCB=(BLKSIZE=4095,RECFM=U)
//SYSUT1 DD DSN=&&TLDRROUT,UNIT=SYSDA,DISP=(OLD,DELETE)
//PRTSUM EXEC PGM=IEBPTCH
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=&&LOADSUM,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSUT2 DD SYSOUT=*,LRECL=133
//SYSIN DD *
          PRINT PREFORM=A
/*

```

TLDR JCL to Load Components to VRDR

The example that follows shows the JCL required to load offline data to the virtual reader (VRDR) using the auxiliary loader offline segment (TLDR).

Notes:

1. *nn* is the release identification and *ssid* is the subsystem ID.
2. This JCL transmits to a data set called **vmnode.vmoid**. Use the ELDRVRDR EXEC to convert this data set to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

```

//TLDRVRDR JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A
//TLDR EXEC PGM=TPFLDRnn,REGION=7000K,PARM='TLDR,SYS=ACP,CLMSIZE=4000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//SALTB DD DSN=ACP.SALTB.LINK.RELnn.ssid,DISP=SHR
//OBJDD DD DSN=ACP.OBJ.LINK.RELnn.ssid,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//ALDRCPD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//TLDRROUT DD DSN=&&VRDRROUT,DISP=(NEW,PASS),UNIT=SYSDA
//TAPEOUT DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
//TEMPLOAD DD DSN=&&SYSUT1,SPACE=(TRK,(20,20)),UNIT=SYSDA
//LOADLIST DD SYSOUT=*
//LOADSUM DD DSN=&&LOADSUM,DISP=(NEW,PASS),UNIT=SYSDA,
//          LRECL=133,SPACE=(TRK,(10,10)),RECFM=FBA
//SYSUDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
/*
//PRTSUM EXEC PGM=IEBPTCH
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=&&LOADSUM,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSUT2 DD SYSOUT=*,LRECL=133
//SYSIN DD *
          PRINT PREFORM=A
/*
//TRANSMIT EXEC PGM=IKJEFT01,

```



```
// PARM='TRANSMIT vmnode.vmuId DDNAME(SYSTSIN) NOLOG NONOTIFY SEQ'
//SYSTSIN DD UNIT=SYSDA,
//          DSN=&&VRDROUT,DISP=(OLD,KEEP)
//SYSTSPRT DD DUMMY
```

OLDR JCL to Load E-Type Programs to GDS

The example that follows shows the JCL required to load offline programs to the general data set (GDS) using the E-type loader offline segment (OLDR).

Note: *nn* is the release identification and *ssid* is the subsystem ID.

```
//OLDRGDS JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A,REGION=2100K,
//          USER=mvsuser,PASSWORD=password
//NAME EXEC PGM=TPFLDRnn,REGION=9000K,PARM='OLDR,CLMSIZE=9000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//OBJLIB DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//CPRTEMP DD UNIT=SYSDA,
//          DSN=&&CPTMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//PROGTEMP DD UNIT=SYSDA,
//          DSN=&&PRTEMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//OUTPUT DD UNIT=SYSDA,VOLUME=SER=dasd,
//          DSN=mvsuser.OUTPUT,SPACE=(TRK,(40,20)),
//          DCB=(RECFM=F,BLKSIZE=4095,LRECL=4095),
//          DISP=(OLD,KEEP,KEEP)
//SALTB DD DSN=ACP.SALTB.LINK.RELnn.ssid,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//PRINTER DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
```

OLDR JCL to Load E-Type Programs to Tape

The example that follows shows the JCL required to load offline programs to tape using the E-type loader offline segment (OLDR).

Note: *nn* is the release identification and *ssid* is the subsystem ID.

```
//OLDRVTAP JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A,REGION=2100K
//NAME EXEC PGM=TPFLDRnn,REGION=9000K,PARM='OLDR,CLMSIZE=9000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.ssid,DISP=SHR
//OBJLIB DD DD DSN=ACP.OBJ.RELnn.ssid,DISP=SHR
//CPRTEMP DD UNIT=SYSDA,
//          DSN=&&CPTMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//PROGTEMP DD UNIT=SYSDA,
//          DSN=&&PRTEMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//OUTPUT DD DSN=OLD.TAPE,UNIT=3480,DISP=(NEW,KEEP),
//          VOL=SER=vvvvvv,LABEL=(,SL),
```

```

//          DCB=(BLKSIZE=4095,RECFM=U)
//SALTBL   DD  DSN=ACP.SALTBL.RELnn.ssid,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//PRINTER  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSIN    DD  *
*** INPUT CONTROL CARDS ARE PLACED HERE

```

OLDR JCL to Load E-Type Programs to VRDR

The example that follows shows the JCL required to load offline programs to the virtual reader (VRDR) using the E-type loader offline segment (OLDR).

Notes:

1. *nn* is the release identification and *ssid* is the subsystem ID.
2. This JCL transmits to a data set called **vmnode.vmoid**. Use the ELDRVRDR EXEC to convert this data set to a spool file that supports virtual reader input. A sample ELDRVRDR EXEC is shipped as segment UELV.

```

//OLDRVRDR JOB MSGLEVEL=1,CLASS=A,MSGCLASS=A,REGION=2100K
//NAME EXEC PGM=TPFLDRnn,REGION=9000K,PARM='OLDR,CLMSIZE=9000000'
//STEPLIB DD DSN=ACP.LINK.RELnn.BSS,DISP=SHR
//          DD DSN=SYS1.EDC.SEDCLINK,DISP=SHR
//          DD DSN=SYS1.PLI.SIBMLINK,DISP=SHR
//LOADMOD DD DSN=ACP.LINK.RELnn.BSS,DISP=SHR
//OBJLIB DD DSN=ACP.OBJ.RELnn.BSS,DISP=SHR
//CPRTMP DD UNIT=SYSDA,
//          DSN=&&CPRTMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//PROGTEMP DD UNIT=SYSDA,
//          DSN=&&PRTEMP,SPACE=(TRK,(100,20)),
//          DCB=(RECFM=FB,BLKSIZE=4095,LRECL=4095),
//          DISP=(NEW,DELETE)
//OUTPUT DD DSN=&&VRDROUT,DISP=(NEW,PASS),UNIT=SYSDA,
//          DCB=(RECFM=F,BLKSIZE=4095,LRECL=4095)
//SALTBL DD DSN=ACP.SALTBL.RELnn.BSS,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//PRINTER DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSIN DD *
*** INPUT CONTROL CARDS ARE PLACED HERE
//*
//TRANSMIT EXEC PGM=IKJEFT01,
// PARM='TRANSMIT vmnode.vmoid DDNAME(SYSTSIN) NOLOG NONOTIFY SEQ'
//SYSTSIN DD UNIT=SYSDA,
//          DSN=&&VRDROUT,DISP=(OLD,KEEP)
//SYSTSPRT DD DUMMY

```

Index

Special Characters

setenv function 149

Numerics

24-bit mode 401, 403

31-bit mode 401, 403

A

addressing mode

24-bit 401

31-bit 401

specifying 401

using the MODE statement 401

ALDR Path card 337, 357

allocating programs

adding function switches 406

in an active TPF system 369

using the offline SALO program 402

allocating spare program slots 405

allocating transfer vectors 404

allocation error listings 407

allocation reports 407

Application Timeout (UCCAPL) 10

auxiliary load

offline input 342

auxiliary loader 341

C

C function trace 98

trace area initialization 105

trace environment customization 100

trace user data 98

C language programs

allocating in 31-bit mode 403

C trace environment customization (UCCCDEB) 100

C trace environment initialization (UCCCEXP) 105

C trace user data (UCCCTRC) 98

category change 213

CCCTIN exit points 22, 24

CCUEXT CSECT 3

chain release 261

CIMR (core image restart area) components

changing 322

copying individual system components 342

core block management

get block with ECB user exit 56

get block without ECB user exit 57

get common block with ECB user exit 58

get system work block 59

release block with ECB user exit 71

release block without ECB user exit 72

release common block with ECB user exit 73

release system work block 74

core load mode number

See load mode number

core resident programs

allocating 402

selective loading during restart 197

CPU external interrupts 29

CPU timer interrupts 30

Create macro processing 31

critical record filing processing 33

CUSR 4

D

data loader 340

data sets

partitioned 407

database reorganization 261

DCTRIT macro 402

deadlock detection user exit 130

debugger trace selection 34, 35

demand counter

location of 402

disk space

reclaiming 371

dynamic exit points

changing to nondynamic 3

dynamic load module (DLM)

environment initialization 39

external function call entry 42

external function call exit 41

return processing entry 45

return processing exit 43

E

E-type loader

changing threshold values 371

clearing file resident records 371

general information 372

offline JCL cards 362

ECB

displaying status 371

ECB (entry control block)

creating 46

enterprise-specific MIB retrieval 202

environment variables 149

error listings

system allocator (SALO) 407

exit points

changing 3

extended globals 304

F

FARF address generation (UCCFSP) 50

FC0TB macro 402

- file address decoding
 - FARF address generation 50
 - split access user exit 81
 - split chain header access user exit 82
- file resident programs
 - allocating 402
- function switches
 - adding 406
 - adding your own 406
 - definition of 406

G

- GAT (global attribute table) 244
- GAT (Global Attribute Table) 248
- general file
 - formatting 322
 - initializing 322
 - IPLing 339
- general file contents 321
- general file load deck
 - creating 322
- general file loader 322
- general postinterrupt processing (UCCGPI) 55
- GENSIP macro 406
- GL0BP 261
- global area 243
- global areas
 - application use of 255
 - area 1 245
 - area 2 247
 - area 3 247
 - area 4 247
 - layout 245
 - locating in a dump 307
 - MDBF layout 252
 - number 304
 - structure 244
- global attribute table (GAT) 244
- Global Attribute Table (GAT) 248
- global block DSECTs
 - considerations when changing 259
 - in global area 1 259
 - list and description 249
 - subsystem design considerations 252
- global directory
 - definition 243
- GL0BA
 - as a part of global area 1 245, 258, 259
 - called by GLOBZ 256
- GL0BY
 - as a part of global area 3 247, 258, 260
 - called by GLOBZ 256
- global environment lists 149
- global fields
 - definition of 243
 - in global area 1 259
 - in global area 3 260
 - SSU common or unique 259

- global macros
 - FILKW
 - description 257
 - example 257
 - updating global fields and records 258, 317, 318
 - GL0BA 248
 - GL0BY 248
 - GLMOD 258
 - description 256
 - updating global fields and records 258, 317, 318
 - GLOBZ
 - addressing global area 1 259
 - addressing global area 3 260
 - addressing global areas 1 and 3 248
 - description 255
 - example 250, 257
 - updating for GL0BA or GL0BY changes 258
 - using to address global fields 250
 - KEYRC 258
 - SYNCC 307, 317
 - description 258
 - updating global fields and records 317, 318
- global record 247, 258
- global storage allocator record (GOA) 244, 261
- global synchronization 255, 306
- global terminology 243
- global variable symbols
 - cross-referencing to PDS members 409
 - finding in a PDS 409
- globals
 - defining addressability
 - See PARM
 - directory slots 248
 - extended 304
 - I-stream unique/shared records 243
 - loading 261, 302
 - programming considerations 258
 - record concatenation 305
 - SSU unique/common 243
- GOA (global storage allocator record) 244, 261
- GOA list entry
 - attribute byte 304
 - directory slot number 305
 - global area number 304
 - number of bytes to strip 304
 - number of doublewords 304
 - ordinal number 305

H

- header label routines 103

I

- I-stream unique/shared records 253, 309
- IDOTB macro 4
- images
 - creating 341
 - defining 342
 - enabling 361
 - IPLing 362

INDSN macro 401, 406
initialization processing 22, 24
input decks, SALO 400
IPLing an image 362

K

key-protected memory 402
key-protected storage 403
keypoint data
 lost 350
keypointable global records 306
keypointing global records 248, 260
keypoints
 moving to the working area 361

L

library function return (ISO-C) 61
load mode number 302
Loader Control cards
 Call General File Keypoint card 327
 Call Keypoint card 350
 Call Online Keypoint card 329
 Call Program card 332, 352, 366
 CC card 326
 Comment card 326
 ELDR Clear card 330, 348
 Image Clear card 326
 LDT card 337, 357
 Load ACPL card 332, 353
 Load AP card 332, 352
 Load Control Program card 328, 349
 Load CTKX card 327, 349
 Load FCTB card 333, 354
 Load ICDF card 333, 353
 Load IPAT card 328, 349
 Load IPLA card 335, 355
 Load IPLB card 336, 356
 Load Keypoint card 329, 349
 Load OPL card 331, 351
 Load PARS card 351
 Load PARS Card 331
 Load RIAT card 336, 356
 Load SIGT card 334, 354
 Load USR1 card 335, 355
 Load USR2 card 335, 355
 Loadset card 366
 Patch cards 367
 Patch Keypoint card 351
 PROG-MOD-BASE Clear card 330, 347
 Program Allocation Table card 365
 Subsystem ID card 348, 365
 System Allocator card 348, 366
loaders
 general information 321
loading E-type programs 362, 367
LODIC Macro (UCCLODC) 62
LU-LU sessions
 accepting 372
 activating 368, 369

LU-LU sessions (*continued*)
 deactivating 370
 deleting from the system 370
 displaying information about 370
 excluding programs 370
 loading 368
 reincluding programs 370
 selecting an RTP connection 189
 using 369

M

macro authorization
 on program allocation 403
macro cross-reference programs
 DCRS cross-reference generation
 attention messages 384
 control cards 383
 description 381
 error messages 385
 JCL 383
 search parameters 381
 DREF report generation
 description 381
 error messages 386
 report heading parameter 382
macro servicing 8
macros
 DCTRIT 402
 FC0TB 402
 GENSIP 406
 IDOTB 4
 INDSN 401, 406
 SI0GT 402
 SSDEF 404
 UXCMC 4
 UXGPIR 4
 UXITC 1
main storage resident programs
 allocating 402
manager validation, SNMP 203
MDBF (multiple database function) 250
memory
 key-protected 402
 page-protected 402
MIB retrieval 202
MODE statement
 specifying 401
movable VIPA processor deactivation 239
Multiple Assembly/Compilation Print Program 377
Multiple Assembly/Compilation Program
 execution parameters 389
 files 387
 input 387
 user considerations 392
multiple database function (MDBF) 250
multiple images 373

N

nondynamic exit points
 changing to dynamic 3

O

object libraries 324, 345, 364
online mini dump user exit 64
online modules
 formatting 322

P

page-protected memory 402
partitioned data sets 407
patching loader offline segment 326
Path card, ALDR 337, 357
PDS (partitioned data set)
 cross-referencing global variable symbols to PDS
 members 409
 searching for a program 409
 searching for global variable symbols 409
PER exit 66
PER2 exit 67
pilot tape 261
pools
 allocating 406
 defining 405
primary globals 243
prime GOA 261, 313
private programs 403
program allocation
 adding function switches 406
program allocation table
 creation of 400
program loadsets
 loading 368
programs
 accepting a loadset 372
 activating 369
 activating selectively 369
 allocating 369, 400
 changing allocation characteristics 371
 deactivating a loadset 370
 deleting a loadset 370
 excluding from a loadset 370
 finding in a PDS 409
 loading 362
 loading to a storage medium 367
 loading to GDS 419
 loading to tape 419
 loading to VRDR 420
 reincluding in a loadset 370

R

rapid transport protocol connections 189
RCS thresholding processing 69
record concatenation 305
record ID attribute table 373

recovery
 catastrophic 20
Resource Control control user exits
 LODIC Macro 62
 suspend ECB 85
 Suspend List Post-Interrupt 86
 suspend list resource checking 87
 TMSLC Macro 93
retrieving enterprise-specific MIB 202
RIAT (record ID attribute table) 373
RIAT processing 75
ROUTC processing 77

S

SAL reports 407
sample STC input 314
SBDTAC SYSTC tag 138
SENDC processing 77
SI0GT macro 402
SIGT (system interprocessor global table) 244
Simple Network Management Protocol (SNMP)
 manager 203
SNMP enterprise-specific MIB retrieval 202
SNMP manager validation 203
spare program slots
 allocating 405
split access (UCCFSC) 81
split chain header access (UCCFHD) 82
SSDEF macro 404
SSU (subsystem user) table 244
SSU table (Subsystem User table) 307
SSU unique/common 252, 306
stack overflow processing entry (ISO-C) 84
stack overflow processing exit (ISO-C) 83
static override bitmap table 4
STC (system test compiler) 261
storage
 key-protected 403
 unprotected 403
 unprotected working 403
storage protection 245
super GOA 244, 261, 262, 301
Super GOA 312
Suspend list PI (UCCSUSE) 85
suspend list PI (UCCSUSP) 86
Suspend list Resource Checking (UCCSUSC) 87
SVC macro decoder 90
SVC macro service services 8
SYNCC macro coding sample 318
synchronizable global field record (SGFR) 307
system allocator (SALO)
 changes to 400
 description 399
 input decks 400
 running the program 406
system allocator table 323, 344
system components
 copying individual components 342
 loading to a storage medium 358
 loading to a TPF image 341

- system components (*continued*)
 - loading to GDS 417
 - loading to tape 417
 - loading to the general file 322, 338
 - loading to VRDR 418
- system data loader 261
- system error user exits
 - catastrophic recovery 20
 - dump override 38, 64
 - SERRC errors 91, 92
 - SNAPC errors 79, 80
- system generation 322
- system interprocessor global table (SIGT) 244
- system resources
 - reclaiming 371
- system test compiler (STC) 261

T

- tape category change 213
- tape volume validation 213
- TC (tightly coupled)
 - See tightly coupled (TC)
- TCP/IP user exits
 - See user exits
- tightly coupled (TC) 252
- TMSLC Macro (UCCTMSL) 93
- TPF macro processing restrictions 8
- TPF transaction services exit 101
- TPFAR exits 95
- TPFDF user exits
 - TPFDF macro trace call 96
 - TPFDF macro trace return 97
- transaction log write routine 101
- transfer vectors
 - allocating 404
 - definition of 404

U

- UCCAPL user exit 10
- UCCCAT user exit 20
- UCCCDBI user exit 12
- UCCCDBR user exit 14
- UCCCDBTS user exit 34, 35
- UCCCDEB user exit 100
- UCCCENV user exit 39
- UCCCEXP user exit 105
- UCCCFB user exit 11
- UCCCFE user exit 15
- UCCCFR user exit 16
- UCCCLE user exit 60
- UCCCLX user exit 61
- UCCCMCP user exit 31
- UCCCMPI user exit 32
- UCCCMXF user exit 25
- UCCCPFI user exit 29, 30
- UCCCREB user exit 27
- UCCCRI user exit 33
- UCCCSER user exit 19
- UCCCSK user exit 17

- UCCCSOE user exit 84
- UCCCSOX user exit 83
- UCCCTRC user exit 98
- UCCDFFC user exit 96
- UCCDFFR user exit 97
- UCCDOT user exit 38
- UCCECB user exit 46
- UCCEFCE user exit 42
- UCCEFCX user exit 41
- UCCENTD user exit 36, 48
- UCCENTN user exit 36, 48
- UCCENTR user exit 36, 48
- UCCEXI user exit 49
- UCCFHD user exit 81, 82
- UCCFLM user exit 52
- UCCFLX user exit 51
- UCCFREB user exit 53
- UCCFSP user exit 50
- UCCGBC user exit 58
- UCCGBE user exit 56
- UCCGBK user exit 57
- UCCGPFA user exit 65
- UCCGSB user exit 59
- UCCLDD user exit 102
- UCCLODC user exit 62
- UCCOMD user exit 64
- UCCPER user exit 66
- UCCPER2 user exit 67
- UCCRBE user exit 71
- UCCRBK user exit 72
- UCCRCB user exit 73
- UCCRIT user exit 75
- UCCRSB user exit 74
- UCCRTC user exit 77
- UCCRTNE user exit 45
- UCCRTNX user exit 43
- UCCSER user exit 92
- UCCSNP user exit 80
- UCCSPX user exit 79
- UCCSRX user exit 91
- UCCSUSC user exit 87
- UCCSUSE user exit 85
- UCCSUSP user exit 86
- UCCSVC user exit 90
- UCCSVW user exit 89
- UCCSVX user exit 88
- UCCTHR user exit 69
- UCCTMSL user exit 93
- UCCUHL user exit 103
- UCCVTO user exit 106
- UCCWAI user exit 107
- UCCWLOG user exit 101
- UCCWTOP user exit 108
- unprotected storage 403
- unprotected working storage 403
- update tape display 102
- user exit control list (UCL) 3
- user exits
 - control program exits, by name
 - CCCPSE, exit point DOT 38
 - CCCPSE, exit point OMD 64

user exits (continued)

control program exits, by name (continued)

CCCPSE, exit point PER 66
 CCCPSE, exit point PER2 67
 CCCPSE, exit point SER 92
 CCCPSE, exit point SNP 80
 CCCPSE, exit point SPX 79
 CCCPSE, exit point SRX 91
 CCCPSF, exit point CAT 20
 CCCTIN exit points 22, 24
 CCENBK, exit point CFB 11
 CCENBK, exit point CREB 27
 CCENBK, exit point DBTS 34, 35
 CCENBK, exit point EFCE 42
 CCENBK, exit point ENTD 36, 48
 CCENBK, exit point ENTN 36, 48
 CCENBK, exit point ENTR 36, 48
 CCENBK, exit point FREB 53
 CCENBK, exit point RTNX 43
 CCNUCL, exit point CMCP 31
 CCNUCL, exit point CMPI 32
 CCNUCL, exit point CMXF 25
 CCNUCL, exit point CPTI 29
 CCNUCL, exit point CPTI 30
 CCNUCL, exit point ECB 10, 46, 62, 93
 CCNUCL, exit point FLM 52
 CCNUCL, exit point FLX 51
 CCNUCL, exit point SVW 89
 CCNUCL, exit point SVX 88
 CCNUCL, exit point WAI 107
 CCSONP, exit point GPFA 65
 CCTLOG, exit point WLOG 101
 CCUEXT, exit point GPI 55
 CLHV, exit point GBE 56
 CLHV, exit point GBK 57
 CLHV, exit point GCB 58
 CLHV, exit point GSB 59
 CLHV, exit point RBE 71
 CLHV, exit point RBK 72
 CLHV, exit point RCB 73
 CLHV, exit point RSB 74
 CPLX, exit point CDBI 12
 CPLX, exit point CDBR 14
 CSTRTD, exit point EFCX 41
 CSTRTD, exit point RTNE 45
 CTR0, exit point CDEB 100
 CTR0, exit point CEXP 105
 CUSR, exit point DFFC 96
 CUSR, exit point DFFR 97

control program exits, by subject

BACKC macro entry (UCCCFB) 11
 C debugger hooks 14
 C debugger initialization 12
 C library function call 15
 C library function return 16
 C stack exception routine 17
 C stack exception routine return 19
 C trace environment customization 100
 C trace environment initialization 105
 C trace user data 98
 catastrophic recovery (UCCCAT) 20

user exits (continued)

control program exits, by subject (continued)

control transfer execution (UCCCMXF) 25
 core-resident enter/back macro (UCCCREB) 27
 CPU external interrupt exit point (UCCCPTE) 29
 CPU timer interrupt exit point (UCCCPTI) 30
 create macro control point (UCCCMCP) 31
 Create macro execution (UCCCMPI) 32
 critical record filing exit point (UCCCRI) 33
 dump override table (UCCDOT) 38
 dynamic load module environment
 initialization 39
 dynamic load module external function call
 (UCCEFCE) 42
 dynamic load module external function call exit
 (UCCEFCX) 41
 dynamic load module return processing entry
 (UCCRTNE) 45
 dynamic load module return processing exit
 (UCCRTNX) 43
 ENTDC macro (UCCENTD) 36, 48
 ENTNC macro (UCCENTN) 36, 48
 ENTRC macro (UCCENTR) 36, 48
 EXITC exit processing (UCCEXI) 49
 fast link macro decoder (UCCFLM) 52
 fast link macro exit (UCCFLX) 51
 file resident enter/back macro (UCCFREB) 53
 general post-interrupt routine (UXGPIR) 4
 get block 56, 57
 get common block 58
 get system work block 59
 library function call (ISO-C) 60
 library function return (ISO-C) 61
 PER exit point (UCCPER) 66
 PER2 exit point (UCCPER2) 67
 pool address retrieval exit point (UCCGPFA) 65
 RCS I/O queue thresholding exit point
 (UCCTHR) 69
 release block 71, 72
 release common block 73
 release system work block 74
 RIAT exit point (UCCRIT) 75
 ROUTC exit point (UCCRTC) 77
 SNAPC error (UCCSNP) 80
 SNAPC error exit (UCCSPX) 79
 stack overflow processing entry 84
 stack overflow processing exit 83
 SVC macro exit 89
 SVC macro exit (UCCSVX) 88
 system error entry (UCCSER) 92
 system error exit (UCCSRX) 91
 TPFDF macro trace call 96
 TPFDF macro trace return 97
 transaction log write routine 101
 update tape message exit routines
 (UCCLDD) 102
 user header label exit routines (UCCUHL) 103
 validate output tape exit routines (UCCVTO) 106
 WAITC macro (UCCWAI) 107
 WTOPC message translation (UCCWTOP) 108

user exits *(continued)*

control program exits, general information

CCUEXT 3
 changing dynamic exit points to nondynamic 3
 changing nondynamic exit points to dynamic 3
 commands 3
 common entry conditions 6
 common programming considerations 7
 common return conditions 7
 creating 5
 installing 4
 performance considerations 1
 UXCMC macro routine 4

E-type program exits, by segment

BK0UX 111
 BRU1 177
 BRU2 178
 BRU3 198
 C542 204
 CDBPUX 133
 CDBUXT 226
 CDC1 124
 CDC2 123
 CDLY 139
 CIM2 199
 CL99 151
 CLA4 170
 CLCH 205
 CLCI 208
 CLCM 169
 CLCQ 167
 CLCS 206
 CLCU 168
 CLCV 207
 CLCX 209
 CLUD 130
 CMVU 201
 COBC 165
 CPSD 210
 CPSU 137, 227
 CSJV 200
 CSLJ 242
 CSXA 153
 CSXB 153
 CSXC 153
 CUIA 225
 CUIR 224
 CUIT 221
 CUIV 223
 CUIW 216
 CVFX 238
 CZ1UX 111
 GCALX 119
 GCLKX 119
 GDATX 119
 UACC 214
 UACP 125
 UALS 188
 UAPN 186
 UARG 152
 UBDB 126

user exits *(continued)*

E-type program exits, by segment *(continued)*

UBOT 215
 UCLB 197
 UCOM 203
 UCPY 166
 UCS1 122
 UCST 121
 UDE0 142
 UDNSEX 136
 UDRR 228
 UDRS 229
 UEL1 134
 UELB 240
 UELC 230
 UELD 120
 UELE 235
 UELF 129
 UELG 135
 UELH 192
 UELI 143
 UELL 150
 UELN 196
 UELR 118
 UELU 194
 UELW 128
 UELX 195
 UENV 149
 UGST 231
 UMIB 202
 UOP1 171
 UOP2 172
 UOP3 241
 UPER 173
 UPX0 175
 UPX1 131
 UPX2 148
 UPX3 145
 UPX6 147
 UPX7 144
 URS1 180
 URS2 181
 URTF 189
 USC2 183
 USC3 184
 USC4 185
 USOK 191
 USOT 236
 UVIP 239
 UXTD 211
 UXTEQ 111
 UXTH 233

E-type program exits, by subject

3270 welcome screen 242
 auxiliary loader 143
 clock global update 119
 Common Link Access to Workstation (CLAW)
 device connection 168, 206
 Common Link Access to Workstation (CLAW)
 device failures 167, 169, 208
 common symbol table 121

user exits *(continued)*

E-type program exits, by subject *(continued)*

- communications source common 122
- data macros 111
- database reorganization 126
- deactivation of processor movable VIPA exit 239
- deadlock detection 130
- dump data 137
- dynamic LU 139
- E-type loader 227
- enterprise specific MIB 202
- extra program record report 143
- file system initialization 215
- general file loader 143
- Get a DNS IP address 136
- get global environment lists 149
- log recovery error processing 151
- LU 6.2 153
- manager validation for SNMP 203
- message router user 165
- messages, routing to nonsocket applications 170
- MIBs, SNMP enterprise-specific 202
- module copy selection/validation 166
- nonsocket application activation 167
- Nonsocket application activation 168
- nonsocket application resource cleanup 169
- nonsocket message 170
- output message filtering 171
- processor deactivation movable VIPA exit 239
- program event 173
- rapid transport protocol connection 189
- recoup 177
- recoup restart 179
- screen connection requests 204
- selective recoup 198
- SLC type-B message handler 199
- SNA communications source program 201
- SNMP manager validation 203
- socket application activation 205
- Socket application activation 206
- socket application resource cleanup 208, 209
- socket cycle-up 207
- system error 209, 210
- system error message 210
- system message processor 227
- system restart 180
- system state 182
- tape display setup 211
- TCP/IP 167, 168, 170, 191, 204, 206, 214
- TPF MQSeries 216, 221, 223, 224, 225
- user command processor 227
- user data recovery copy support 229
- user data recovery restore support 228
- user global symbol table 231
- user label routines 233
- user symbol override table 236
- validation of SNMP manager 203
- VFA restart 238
- virtual IP address (VIPA) processor deactivation exit 239
- VisualAge debuggers 133, 226

user exits *(continued)*

E-type program exits, by subject *(continued)*

- WTOPC page control 241
- E-type program exits, general information
 - performance considerations 109
 - user considerations 111
- user exits CSECT (CCUEXT) 3
- UXCMC macro routine 4
- UXGPIR macro routine 4
- UXITC macro routine 1

V

- validate output tapes 106
- validate SNMP manager 203
- variable cross-reference (VCRS) listing program
 - cross-referencing global variable symbols to PDS members 409
- dataset definition statements 410
 - SORTIN 410
 - SORTLIB 410
 - SORTOUT 410
 - SYSPRINT 410
- error messages 414
- finding global variable symbols in a PDS 409
- finding programs in a PDS 409
- PDS option defaults 413
- virtual IP address (VIPA) processor deactivation exit 239

W

- WLOGC write processing 101
- WTOPC message translation 108



File Number: S370/30XX-34
Program Number: 5748-T14



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH31-0149-15

