

Transaction Processing Facility



# System Macros

*Version 4 Release 1*



Transaction Processing Facility



# System Macros

*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 xi.

**Fourteenth Edition (June 2002)**

This is a major revision of, and obsoletes, SH31-0151-12 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

<b>Tables</b> . . . . .	ix
<b>Notices</b> . . . . .	xi
Trademarks . . . . .	xi
<b>About This Book</b> . . . . .	xiii
Who Should Read This Book . . . . .	xiii
How This Book Is Organized . . . . .	xiii
Conventions Used in the TPF Library . . . . .	xiv
How to Read the Syntax Diagrams . . . . .	xv
Related Information . . . . .	xviii
IBM Transaction Processing Facility (TPF) 4.1 Books . . . . .	xviii
Miscellaneous IBM Books . . . . .	xviii
Online Information . . . . .	xviii
How to Send Your Comments . . . . .	xix
<b>System Macros Introduction</b> . . . . .	1
Control Program (CP) Linkage . . . . .	1
Overview . . . . .	1
Register Defaults . . . . .	2
The Stacks . . . . .	3
Linkage . . . . .	4
Sample Stack Area Use . . . . .	6
Coding Example for Control Program (CP) Linkage . . . . .	6
Stack Definition Example . . . . .	8
<b>System Macros</b> . . . . .	11
\$ADPC—Add Work to a List on Specified I-Stream . . . . .	12
\$CKMAC—Validate Use of Restricted Macro . . . . .	14
\$CONBC—Connect Block to ECB Virtual Memory . . . . .	16
\$CPUC—Interface for SIGP Services . . . . .	19
\$CRISC—Cross Over to Another I-Stream . . . . .	22
\$DCOLC—Data Collection Hook Insertion . . . . .	24
\$DISBC—Disconnect A Block from the ECB Virtual Memory . . . . .	27
\$FINDC—CP Find A File Record . . . . .	29
\$FORKC—Create an Asynchronous ECB . . . . .	32
\$GCOMC—Obtain Common Storage Block . . . . .	33
\$GETBC—Obtain Storage Block . . . . .	35
\$GETRC—Get a Control Record . . . . .	38
\$GEVAC—Convert SVM Address to EVM Address . . . . .	40
\$GIOBC—Get Available I/O Control Block Address . . . . .	42
\$GMNBC—Get Contiguous EVM Storage . . . . .	44
\$GSVAC—Convert EVM Address to SVM Address . . . . .	46
\$GSWBC—Get System Work Block Address . . . . .	48
\$GSYSC—Get System Heap Storage . . . . .	50
\$GTSTC—Get Contiguous EVM Stack Storage . . . . .	53
\$LCKRC—Lock a Virtual File Access (VFA) Shared Lock or Exclusive Lock . . . . .	55
\$LOCKC—Lock a Resource . . . . .	56
\$MASKC—Change the System Mask . . . . .	58
\$MONTC—Set Supervisor State (Monitor Mode) with PSW Return . . . . .	61
\$MOVEC—Move Data Between EVM and SVM . . . . .	62
\$RCOMC—Release Common Storage Block . . . . .	64
\$RECV C—Recover from Program Check . . . . .	66

\$RELBC—Release Storage Block . . . . .	68
\$RELRC – Release a Control Record . . . . .	70
\$RETRC – Retrieve or Modify a Control Record. . . . .	72
\$RIOBC—Release Input/Output Control Block (IOCB) Address . . . . .	75
\$RMNBC—Release Acquired Storage . . . . .	77
\$RSWBC—Release System Work Block . . . . .	79
\$RSYSC—Release System Heap Storage . . . . .	81
\$SWSPC—Switch Address Space . . . . .	84
\$TCPLC—Control Program (CP) Tape Logging . . . . .	87
\$ULKRC—Unlock a Virtual File Access (VFA) or Record Hold Table (RHT) Lock . . . . .	90
\$UNLKC—Unlock a Resource. . . . .	91
\$VALEC—Validate Entry Control Block (ECB) Virtual Memory (EVM) Address . . . . .	93
#SBRC—Standard Linkage Macro Subroutine . . . . .	94
ADDFC—Add a Block to the Top of a Dispatch List . . . . .	98
ADDLC—Add Block to the End of a Dispatch List . . . . .	101
BBEWP—Recoup Error Item Setup . . . . .	104
BBPDH—Recoup Record Find and Count Interface . . . . .	106
BBWRT—Recoup Logging Item Setup . . . . .	108
BRPRO—Query Recoup Options . . . . .	110
BRSTR—Recoup Register and Entry Control Block (ECB) Work Area Restore . . . . .	113
BSAVE—Recoup Register and Work Area Save. . . . .	115
BSCQC—Release Core Blocks That Are Not Attached to an ECB . . . . .	117
BSTAK—Recoup Stack. . . . .	118
BSYNC—Recopy SYNCC Facility . . . . .	120
CEBIC—Change MDBF Subsystem/Subsystem User ID . . . . .	122
CFCONC—Connect to a Coupling Facility List or Cache Structure . . . . .	128
CFDISC—Disconnect from a Coupling Facility List or Cache Structure . . . . .	143
CFISVC—Find Entry in the Macro Information Tables . . . . .	146
CFRQC—Coupling Facility Request . . . . .	148
CFVCTC—Check or Modify a List Notification Vector. . . . .	152
CIOSC—Request a Mount, Dismount or Status of an SDA. . . . .	159
CIOUC—Initialize and Reset Communication Lines . . . . .	161
CLAWCC—CLAW API Linkage . . . . .	164
CLNKC—Control Program (CP) Call and Link . . . . .	166
CPDSC—Generate Control Program (CP) DSECT. . . . .	168
CPLKC – Link to CP Routines. . . . .	169
CPRND—Control Program (CP) Rounding. . . . .	172
CRASC—Send Message to CRAS . . . . .	173
CRATC—Search CRAS Status Table. . . . .	175
CREGPC—Create a Macro Group Definition . . . . .	179
CRESVC—Create an SVC/Fast-Link Table Entry . . . . .	180
CROSC—Cross-Subsystem Access Service Request. . . . .	185
CTKL—SLC Channel Keypoints Setup . . . . .	189
CVTPC—Convert Tape Status Table Pointer . . . . .	197
CWRTC—Write Critical Message to the System Console . . . . .	200
CXFRC—Create a New ECB and Transfer Control . . . . .	204
CYDNC—Cycle Down Utility CP Interface . . . . .	210
DHASHC—Hash Resource Name. . . . .	212
DLCKC—Modify Lock and I/O Interrupt Status . . . . .	214
DLNKC—Define Stack DSECT for Control Program (CP) Routine . . . . .	216
DSDAC—Dismount a Symbolic Device Address (SDA) . . . . .	217
ECBLC—Remove IOBs Associated with an ECB Address . . . . .	218
ELLEC—Schedule an ECB to Exit or Resume . . . . .	219
ENATC—Activate or Deactivate C Function Trace for an ECB . . . . .	221
ESFAC—Obtain Symbolic File Address Information . . . . .	226
FACZC—Compute File Address . . . . .	228

FCTLG–File Control . . . . .	233
FDCTC–File Data Chain Transfer . . . . .	237
FLFAC–Flush a Record from VFA Buffers . . . . .	241
FLSPC–File a Special Record . . . . .	243
FLVFC–Flush a Record from VFA Buffers . . . . .	246
FNSPC–Find a Special Record . . . . .	249
FTSTC–Find/File. . . . .	252
GCFBC–Get Coupling Facility Work Block Address . . . . .	254
GCLAC–Get a Specified CLAW Block Type . . . . .	255
GDSCC–General Data Set (GDS) Control . . . . .	258
GNAMC–Get Program Prolog Area (PPA) Functional Name Information . . . . .	261
GROUP–Recoup Descriptor Record Access. . . . .	264
GRRTC–Get Record Code Check (RCC) Reference Table . . . . .	275
GSVAC–Convert an EVM Address to an SVM Address. . . . .	276
GSWBC–Get a System Work Block (SWB) Address. . . . .	278
GSYSC–Get System Heap Storage . . . . .	279
HIOSC–Halt an I/O Operation . . . . .	281
IBMSVC–Generate IBM SVC and Fast-Link Tables . . . . .	282
ICELOG–TARGET(TPF) C Language Support Epilog . . . . .	283
ICLANC–Call a Secondary Library Routine . . . . .	285
ICPLOG–TARGET(TPF) C Language Support Prolog . . . . .	286
IDATB–Build Selective Memory Dump Table. . . . .	289
IDATG–Generate Selective Memory Dump Table Entry. . . . .	293
IDOTB–Dump Override Table Build . . . . .	296
IFRVTG–Test RID/RVT Address . . . . .	300
IGATC–Get Global Attribute Table Entry . . . . .	302
ILCKCB–Lock a Control Block Area Macro . . . . .	304
INDEX–Recoup Descriptor Record Structure . . . . .	306
IOIRC–Return from CIO Input/Output (I/O) Interrupt Processing . . . . .	323
IPSDC–Call TCP/IP Native Stack Common Service Routine . . . . .	324
IPURGE–Purge Data from Queue . . . . .	325
ISDAC–Interrogate Symbolic Device Address (SDA) Status . . . . .	326
ISNSE–Add an Entry to the Sense Table . . . . .	328
IULKCB–Unlock a Control Block Area . . . . .	330
IVTYPE–GETCC Block Type Verification . . . . .	331
KEYCC–Change Protection Key . . . . .	332
KEYUC–Keypoint Update . . . . .	334
LCPCCC–Low Address Protect Set and Restore. . . . .	335
LEBIC–Load and Shift SS/SSU ID . . . . .	337
LEMIC–Lock Entry Management Interface . . . . .	340
LMONC–Reset Supervisor State (Problem State). . . . .	344
MAXBC–Get Maximum Number of Storage Blocks . . . . .	345
MODEC–Change Addressing Mode . . . . .	347
MONTC–Set Supervisor State (Monitor Mode). . . . .	348
MONWC–Suspend ECB, Pending I/O Completion . . . . .	349
MOVEC–Move Data Between EVM and SVM . . . . .	350
MPIFC–Request MPIF Service . . . . .	352
MSDAC–Mount a Symbolic Device Address (SDA) . . . . .	356
MSPIC–Control MPIF Device . . . . .	358
NUMBC–Query Number of Storage Blocks Available . . . . .	360
NUMLC–Get Count of Blocks Queued on a Dispatch List. . . . .	361
NXTLC–Get Address of Next Block Queued on a Dispatch List. . . . .	363
NXTPC–Chain Chase through Prefix Pages. . . . .	366
NXTRC–Get next TPF Trace Table Entry . . . . .	367
PAUSC–Control System MP Environment . . . . .	369
PERCC–Enable/Disable Program Event Recording (PER) . . . . .	371

PFSWC–Reset Pool Function Switch . . . . .	373
PHYBC–Return Physical Size of Storage Block . . . . .	376
PIOFC–Initiate a Preemptive I/O Request . . . . .	378
PIORC–Return from PIO I/O Interrupt Processing . . . . .	380
PKEYC–Keypoint Communication Data . . . . .	381
PLNAC–Check Symbolic Line Type . . . . .	383
PLNSC–Find SLST Entry . . . . .	386
PLONC–Place on Queue. . . . .	388
PROGC–Return Program Information . . . . .	390
QASNC–Query Asynchronous I/O Event Facility . . . . .	392
QGDSQ–Query General Data Set (GDS) Input/Output (I/O) Queue . . . . .	394
RCFBC–Release Coupling Facility Work Block Address . . . . .	396
RCLAC–Release a Specified CLAW Block Type . . . . .	397
RCRTC–Clean Up Blocks in the CRET . . . . .	399
RCSSC–Access the Record Cache Subsystem Status Table . . . . .	400
RDCTC–3705 Communications . . . . .	403
RESMC–Resume Normal CIO I/O Processing . . . . .	405
RIOSC–Reset an I/O Operation . . . . .	406
RITID–Access RIAT Entry . . . . .	407
RLNKC–Return to CP Calling Routine and Reset Stack Pointer . . . . .	409
RPVRC–Read and Process Program Version Record . . . . .	411
RSWBC–Release a System Work Block (SWB) . . . . .	412
RSYSC–Release System Heap Storage . . . . .	413
RVTCC–Search RVT Entries . . . . .	415
RWGTC–Release a Lock on a WGTA Entry. . . . .	418
SANQC–Define and Enqueue Resource, Signal Aware. . . . .	419
SENDC–Send Message to Terminal. . . . .	422
SETOC–Set Maximum Times to Avoid Application Timeout for an ECB . . . . .	428
SETTC–Set C Function Trace Information for an ECB . . . . .	431
SICFC–IPC Service Request . . . . .	435
SIOSC–Start an Input/Output (I/O) Operation . . . . .	436
SIPCC–System Interprocessor/Inter-I-Stream Communication . . . . .	437
SLCQC–SLC Queue Handling . . . . .	443
SLMTC–Send LMT High Speed Transmission . . . . .	446
SLNKC–Control Program (CP) Save Link Data & Set Stack Pointer . . . . .	448
SNDLC–Send Control Message to 3270 . . . . .	450
SOUTC–Write Path Information Unit (PIU) Systems Network Architecture (SNA) Input/Output (I/O) . . . . .	453
SPNDC–Suspend Normal CIO Processing . . . . .	457
SSMMC–Set System Mask . . . . .	459
STIMC–Time-Initiated CP Routine Execution . . . . .	460
STLUC–Send LU 6.2 Message from OMT . . . . .	462
SWCHC–Set and Test Lethal Utility Switch . . . . .	464
SWISC–Switch Entry to Another I-Stream . . . . .	467
SYCON–System Configuration . . . . .	471
TANCC–Transaction Anchor Table Control (TANC) . . . . .	473
TASBC–Turn Off Time Available Supervisor Switch . . . . .	475
TASTC–Turn on Time Available Supervisor Switch . . . . .	476
TCLAC–Write a CLAW Error Log. . . . .	477
TDCTC–General Tape Data Chain Transfer . . . . .	478
TDTAC–General Tape Data (GDS) Transfer . . . . .	481
TERMC–Kill a Threaded Process . . . . .	485
TIOSC–Test Input/Output (I/O) Service. . . . .	486
TMSLC–Time Slice an ECB. . . . .	488
TMTKC–Get the Unique Token for the Current Transaction . . . . .	492
TPCNC–Tape Control . . . . .	493

TPINC—Special Tape Interface . . . . .	495
TYPBC—Obtain Block Type and Size . . . . .	500
USATC—Create User Storage Allocation Table Entry . . . . .	501
USRSVC—Generate the User SVC Tables . . . . .	504
UXITC—User Exit Interface Linkage . . . . .	506
WLOGC—Write to the Recovery Log. . . . .	508
WRSTC—Get Load Module Writable Static Data Length . . . . .	512
YIELD—Yield Control. . . . .	514
<b>Index . . . . .</b>	<b>517</b>



---

## Tables

1.	Return and Reason Codes for the CFCONC Macro . . . . .	135
2.	Return and Reason Codes for the CFDISC Macro . . . . .	144
3.	Branch Routines for the CFVCTC Macro with REQUEST=MODIFYVECTORSIZE Coded . . . . .	155
4.	Branch Routines for the CFVCTC Macro with REQUEST=LTVCENTRIES Coded . . . . .	156
5.	Branch Routines for the CFVCTC Macro with REQUEST=TESTLISTSTATE Coded . . . . .	157
6.	Specification of the TYPE Parameter for the INDEX Macro and Parameter Requirements . . . . .	306
7.	Storage requirements for the PLNAC Macro . . . . .	384
8.	Time Slice Name Table . . . . .	490



---

## 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:

C/370  
ECKD  
Enterprise Systems Connection Architecture  
ESCON  
IBM  
Language Environment  
MQSeries  
OS/390  
Sysplex Timer

System/370  
VisualAge.

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

---

## About This Book

The macro manuals are the primary references for assembler language macro usage under the TPF system.

*TPF System Macros* contains descriptions of macros to be used only by the TPF system. For general use macros, refer to *TPF General Macros*.

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 is intended for use by system programmers.

---

## How This Book Is Organized

The macro manuals contain the detailed specifications for most TPF macros. Not included are system installation macros, data macros, structured programming macros, and macro groups that are related to specific functions or user tasks.

The first section of *TPF System Macros* contains an introduction to control program considerations such as, linkage, register defaults, and control stacks. The remaining section consists of descriptions for all the system macros. There are four general classifications of system macros:

- Those requiring special authorization
- Those not requiring authorization
- Those to be used in the control program only
- Those restricted to special packages (such as, Recoup).

The interfaces to system macros are not guaranteed. Unlike the general use macros, no effort is made to ensure the interface remains stable. The interface for the system macros may change without notice.

The following macros are described outside this book:

- **General use macros** for application programs are in *TPF General Macros*.
- **Structured programming macros** for application programs are in *TPFDF and TPF Structured Programming Macros*.
- **System Initialization Program (SIP) macros** are in *TPF System Generation*.
- **Program Test Vehicle macros**, used by test driver programs, are in *TPF Program Development Support Reference*.
- **Dump format macros** are in the online segment (ICDF) of the diagnostic output formatter (DOF), described in *TPF Program Development Support Reference*.

An index appears at the end of this book.

## Conventions Used in the TPF Library

The TPF library uses the following conventions:

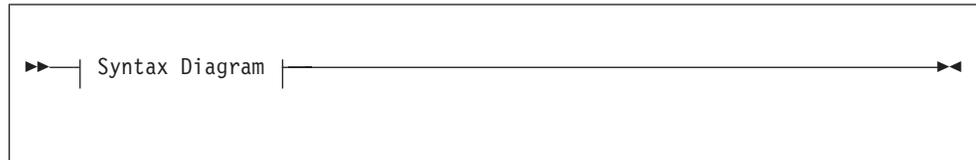
Conventions	Examples of Usage
<i>italic</i>	Used for important words and phrases. For example: A <i>database</i> is a collection of data.  Used to represent variable information. For example: Enter <b>ZFRST STATUS MODULE</b> <i>mod</i> , where <i>mod</i> is the module for which you want status.
<b>bold</b>	Used to represent text that you type. For example: Enter <b>ZNALS HELP</b> to obtain help information for the ZNALS command.  Used to represent variable information in C language. For example: <b>level</b>
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);
<b><i>bold italic</i></b>	Used for emphasis. For example: You <b><i>must</i></b> type this command exactly as shown.
<b><u>Bold underscore</u></b>	Used to indicate the default in a list of options. For example: <b>Keyword=OPTION1   <u>DEFAULT</u></b>
Vertical bar	Used to separate options in a list. (Also referred to as the OR symbol.) For example: <b>Keyword=Option1   Option2</b>  <b>Note:</b> 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= <i>option</i>
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  <b>Notes:</b> 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.

---

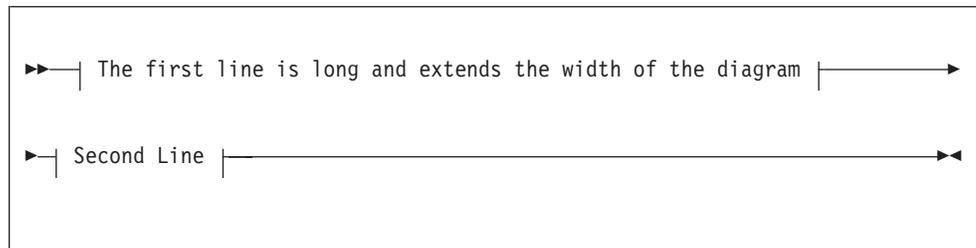
## How to Read the Syntax Diagrams

This section describes how to read the syntax diagrams (informally called *railroad tracks*) used in this book.

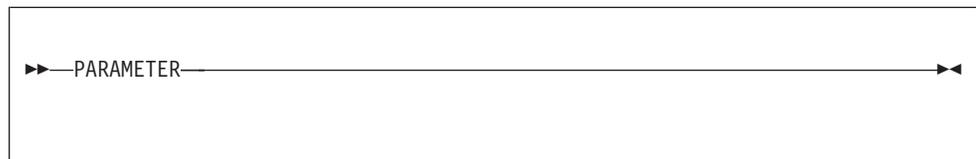
- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads and ends on the right with 2 arrowheads facing each other.



- If a diagram is longer than one line, the first line ends with a single arrowhead and the second line begins with a single arrowhead.

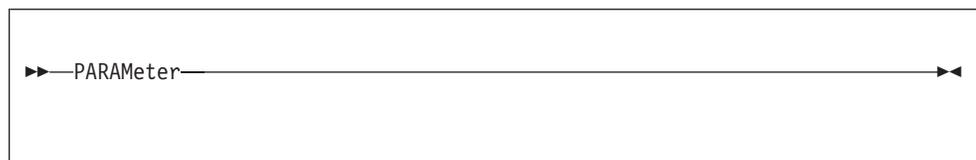


- A word in all uppercase is a parameter that you must spell **exactly** as shown.

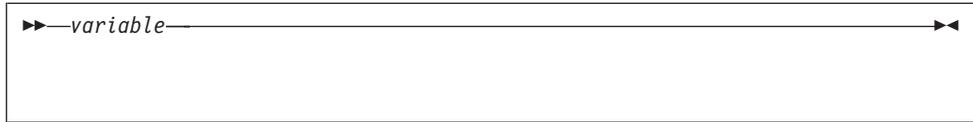


- If you can abbreviate a parameter, the optional part of the parameter is shown in lowercase. (You must type the text that is shown in uppercase. You can type none, one, or more of the letters that are shown in lowercase.)

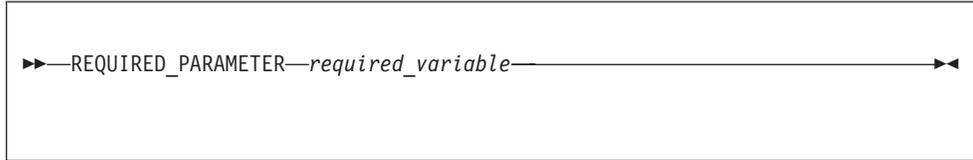
**Note:** Some TPF commands are case-sensitive and contain parameters that must be entered exactly as shown. This information is noted in the description of the appropriate commands.



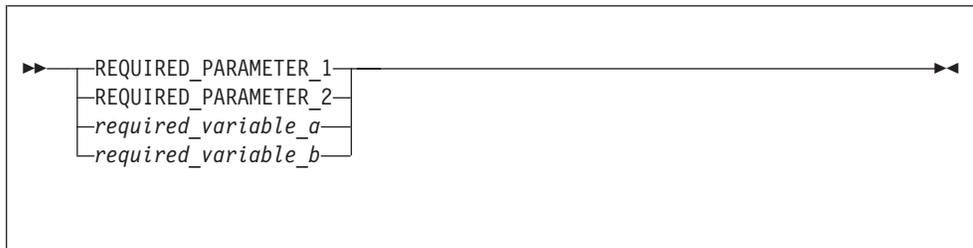
- A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.



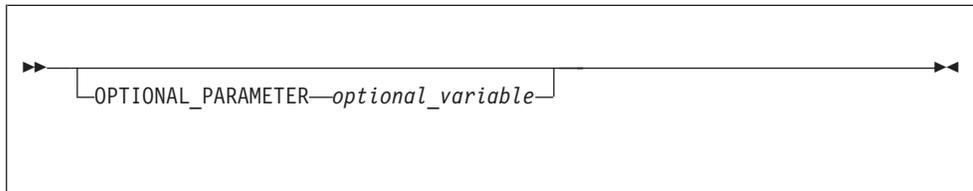
- Required parameters and variables are shown on the main path line. You must code required parameters and variables.



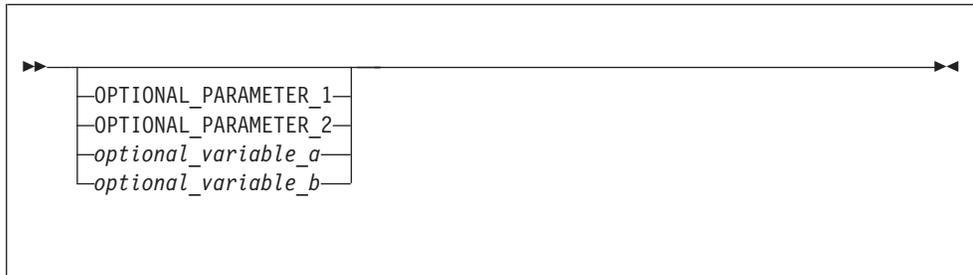
- If there is more than one mutually exclusive required parameter or variable to choose from, they are stacked vertically.



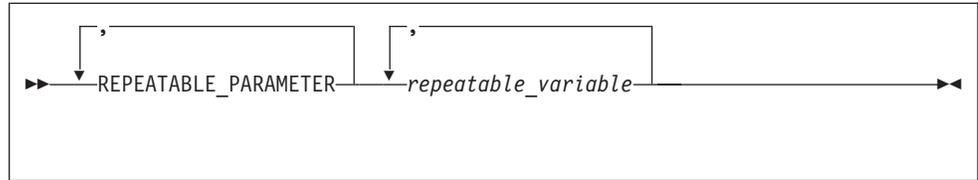
- Optional parameters and variables are shown below the main path line. You can choose not to code optional parameters and variables.



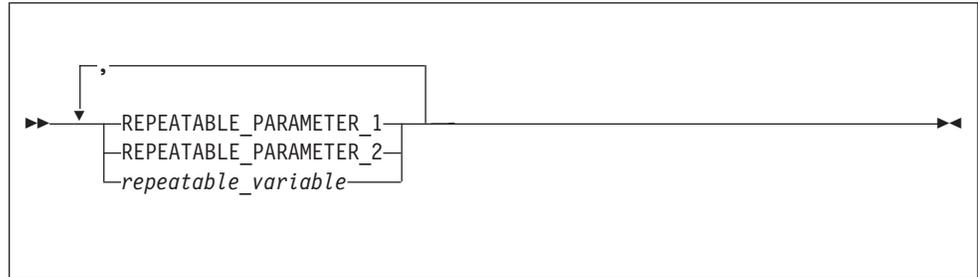
- If there is more than one mutually exclusive optional parameter or variable to choose from, they are stacked vertically below the main path line.



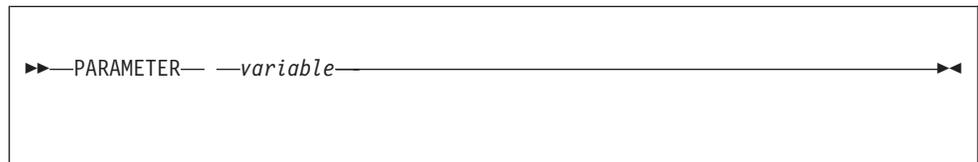
- An arrow returning to the left above a parameter or variable on the main path line means that the parameter or variable can be repeated. The comma (,) means that each parameter or variable must be separated from the next parameter or variable by a comma.



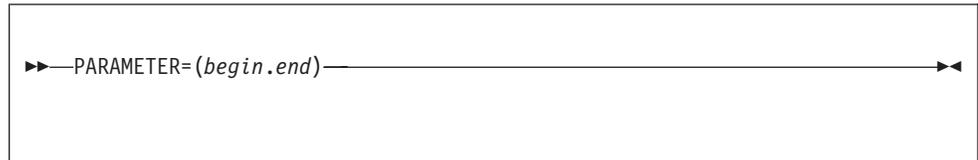
- An arrow returning to the left above a group of parameters or variables means that more than one can be selected, or a single one can be repeated.



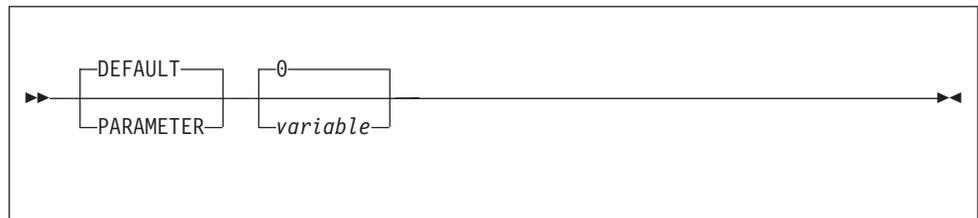
- If a diagram shows a blank space, you must code the blank space as part of the syntax. In the following example, you must code **PARAMETER** *variable*.



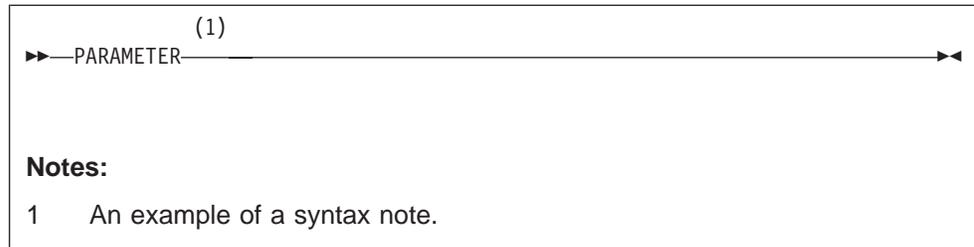
- If a diagram shows a character that is not alphanumeric (such as commas, parentheses, periods, and equal signs), you must code the character as part of the syntax. In the following example, you must code **PARAMETER=(begin.end)**.



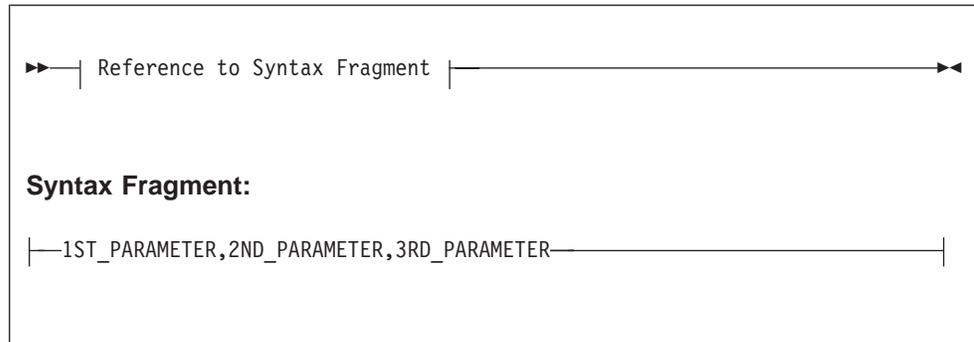
- Default parameters and values are shown above the main path line. The TPF system uses the default if you omit the parameter or value entirely.



- References to syntax notes are shown as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.



- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.




---

## 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 ACF/SNA Data Communications Reference*, SH31-0168
- *TPF Application Programming*, SH31-0132
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF Database Reference*, SH31-0143
- *TPF General Macros*, SH31-0152
- *TPF Operations*, SH31-0162
- *TPF Program Development Support Reference*, SH31-0164
- *TPF Programming Standards*, SH31-0165
- *TPF System Generation*, SH31-0171
- *TPF System Installation Support Reference*, SH31-0149
- *TPFDF and TPF Structured Programming Macros*, SH31-0183

### Miscellaneous IBM Books

- *OS/390 MVS Sysplex Services Guide*, GC28-1771.

### 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](mailto: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



---

## System Macros Introduction

Macros in this publication are restricted to system use and are known as *system macros*. System macros must not be used by application programs. There are 3 types of system macros:

- System macros that require authorization.  
These macros are guarded by the macro checking process. Programs using them must be authorized in the program allocation table (PAT) during installation or by the system operator.
- System macros that do not require authorization.  
There is no mechanism for authorizing use of these macros. They only operate in a system environment.
- Macros restricted to the control program (CP).  
These macros only operate correctly when invoked by the CP. For example, R9 might not point to an entry control block (ECB) (this is a violation of the application interface for E-type programs), reserved hardware control sequences are being executed (operation of hardware is restricted to the CP), or privileged instructions are being processed (also restricted to the CP). Macros restricted to the CP do not require authorization.

Explanations of macro authorization appear in the \$CKMAC macro and in the allocation section of *TPF System Installation Support Reference*. See “\$CKMAC–Validate Use of Restricted Macro” on page 14 for more information about the \$CKMAC macro.

An additional category of macros appears in this publication – those that relate specifically to a particular package and are not usable outside that package.

---

## Control Program (CP) Linkage

This section provides overview information, as well as information about:

- Register defaults
- Stacks
- Linkage
- Sample stack area use
- Coding example for CP linkage
- Stack definition example.

### Overview

The control program (CP) linkage is intended to provide TPF system programmers with a mechanism for linking between control program routines. Included here are a set of rules governing the linkage and a set of system macros that adhere to the rules. This linkage is available for use when modifying existing CP sections or creating new ones.

The tightly-coupled multiprocessing environment requires that certain system macros be coded in every control program CSECT that can be processed from multiple I-streams. The system macros provide linkage conventions and define work areas on I-stream unique stacks that protect the TPF system from corruption due to

multiple I-streams using the same work space. All stack manipulation and CP linkage must be done only by these system macros to provide efficient, cost-effective modification in the future.

The mechanism assumes:

- **The use of I-stream-unique stack areas that are used by all control program routines:** Stack areas can be of any size but must be large enough to include the total work area of the user requiring the largest work area at each level of stack use.
- **Default register usage:** These defaults can be overridden to conform to existing CP routines except for R13 which is always assumed to point to a stack.
- **No controls to monitor stack area overflow or underflow:** For performance reasons, logic and control flow in both the CP and user exits are the only way to monitor stack usage and prevent stack overflow.

Stacks exist for the different environments in which the TPF system operates:

- Input/output (I/O)
- External
- Supervisor call (SVC)
- Error
- Error I/O
- Fast Link
- Machine Check.

You can also define additional stack areas, such as user-defined stack areas, as needed. R13 is initialized to point to each respective stack at entry to the environment. (For example, first level I/O interrupt handler points R13 to the I/O stack; macro decoder points R13 to the SVC stack). Therefore, all CP routines assume R13 addresses a stack save area.

When an environment is entered, a work area, or stack, must be established for that environment. This stack can be of any size but must be large enough to include the total work area of the user requiring the largest work area at each level of stack use. The current implementation of stack control is limited and assumes an implied limit to levels of nested linkage calls because of the predefined stack size.

## Register Defaults

This section provides information about register defaults as follows:

- Register 15 is used as the base address of the executing routine.
- Register 14 points to the next sequential instruction (NSI) of the routine that invoked this routine, that is, it is the return address (link register).
- Register 13 always points to a stack work area.
- Register 0 through Register 12 should be saved and restored over all calls. In general, contiguous registers starting at R0 should be used to pass parameters between calls (all registers should be saved and restored by the called routine with the exception of passed parameters which are not restored). This is to enable a single load multiple instruction to be used to restore registers.

**Note:** The entire CP does not necessarily adhere to these register conventions. It is possible to override each convention, with the exception of R13 usage.

# The Stacks

This section provides information about:

- Initialization
- DCTSTK data area
- User-defined data area.

## Initialization

The system initializer, CCCTIN, carves out a stack for each environment for each I-stream. After initialization, while the TPF system is running, each environment entry point (for example, the macro decoder for SVC and the I/O interrupt handler for I/O) sets R13 to point to the appropriate stack. Therefore, the entire CP can assume R13 always points to a stack entry.

Each stack contains enough space to save registers and work space for the current CP linkage implementation.

## DCTSTK Data Area

The DCTSTK data area that defines stack entries. It has a fixed area that defines space for registers, a performance area, and space for use by macros. In addition, by using the SLNKC macro you can define a variable amount of space on the stack for your use.

All stack manipulation and CP linkage must be done only by the SLNKC, DLNKC, CLNKC, and RLNKC macros to provide simple modification in the future.

**Predefined Equates:** Equate values are used to address predefined areas. The size of this area is constant, making it easier to view in a dump.

**Previous Stack Pointer:** This fullword contains a pointer to the previous stack area used. It is used to return to the caller and is for debugging dumps because it is often necessary to locate the stack area of the caller.

**Register Save:** Sixteen fullwords are allocated to save all general registers.

**Performance Area:** The performance area is allocated in every stack entry for use by performance-oriented routines such as storage block management routines that do not have to subtract and push the stack pointer. These routines cannot afford the added cost of pushing the stack pointer. Routines using the performance area cannot call a lower level routine.

**STKINL Macro Save Area:** The STKINL macro save area is an inline save area used by system macros to temporarily save and restore registers and data over the macro call. This alleviates the problem of finding a place to save registers during macro calls (inline macros or system macros that generate linkage to central routines).

## User-Defined Area

The user-defined area, in DSECT form, can be variable in size depending on routine requirements, single register addressability, and stack size. Single register addressability limits the space to 4096 bytes. This space is defined by coding define storage instructions between the SLNKC and DLNKC macros.

**Note:** These areas are not automatically initialized during subroutine calls and if initialization is important must be handled by the user.

## Linkage

You should always use the SLNKC, DLNKC, CLNKC, and RLNKC system macros, which are CP linkage and stack manipulation macros, when linking between CP routines that require stacks. These system macros use the default register conventions described in “Register Defaults” on page 2, but the defaults can easily be overridden for general TPF system use.

This linkage meets the following requirements:

- The previous stack pointer can always be seen in the current stack entry.
- Minimal instructions are used for the linkage.
- All registers, except the linkage register and parameter registers, can be saved and restored over the macro call.
- Default register conventions can be overridden and, in addition, coexist with different register usage. This means one routine can use the CLNKC macro with one set of link registers while another routine can use the CLNKC macros with other link registers.

The following are assumptions about this linkage:

- The caller must know the base register used by the caller routine. This is because the base register of the caller routine is saved on the stack prior to the call.
- If the default registers for the SLNKC, CLNKC, and RLNKC macros are overridden, the called routine must know which register was used as the return link register of the caller.
- The stack has the following areas:
  - A fixed size area for TPF system use.
  - An optional, variable size area for your use.

These areas do not exceed 4096 bytes of doubleword entries.

- Stack entries are located in contiguous storage.

### Initial Linkage

The CCCTIN system initializer carves out stacks for each environment (for example, SVC, I/O, MCHK, ERROR) for each I-stream and establishes a pointer to each stack in the prefix page. This pointer is initialized to the bottom of the stack minus 1 entry so that the stack is ready to be addressed upon entry to the environment.

When an environment is entered, R13 is set to point to the proper stack for the current I-stream. Since the stack pointer is located in the prefix page, it is addressable by the entire CP and is, by definition, I-stream-unique.

For example, when an SVC interrupt occurs, the macro decoder must:

```
L R13,PFSSAVE
```

This sets R13 to the SVC stack. You can now code the SLNKC and CLNKC macros.

### Calling Linkage

To call a CP routine, the caller's link register or branch-to-register is saved in the register save area on the stack. Then the caller loads the branch-to-register (which will become the called routines base register) and BASRs to the routine. If default register conventions are used:

```

CALLER  ST  R15,STKR15(,R13)
        L   R15,CALLEE
        BASR R14,R15

```

**Note:** The branch-to-register, R15, is saved on the stack before the call to preserve its contents over the macro call. The called routine must not overlay the contents of STKR15 when it saves registers, and in addition it must restore R15 when returning.

### Linkage upon Entry to a Routine

Upon entry to a CP routine that had been called by the calling linkage described previously (see “Initial Linkage” on page 4 for more information) the lowest number register used by the routine through register 15 is stored into the caller’s stack register save area. R13 is then decremented by the size of the work area needed by the current routine and its previous value, which points to the last stack entry, is saved in the current stack entry. If default register conventions are used:

```

CALLEE  STM  R0,R14,STKR0(R13)
        SL   R13,STKLEN
        MVC  STKPREV,STKLEN+STKR13(R13)

```

#### Notes:

1. Remember R15 was saved already by the caller. Therefore, it is essential that the caller and the callee know what is the base register of the called routine.
2. The registers are saved on the stack first and then R13 is pushed. To find the register contents at entry to a routine in a dump, you must first get the pointer to the last stack entry (STKPREV).

### Linkage to Return

To return to a CP routine that was called by the linkage described previously, R13 is restored to point to the caller’s stack area. All registers saved by the called routine (except parameters being passed between calls) are restored, and return is made by a BR R14. If default register conventions are used:

```

L   R13,STKPREV(R13)
LM  R0,R15,STKR0(R13)
BR  R14

```

**Note:** This includes the branch-to-register or BASE previously saved in the calling linkage logic.

### Linkage Notes

Keep in mind the following:

- When performance is not critical, all registers should be saved on the stack (low register of R0). Registers not passing data (as defined by the routine interface and coded by the RLNKC macro) should be restored upon return. This will enhance maintainability and debugging ability.
- Generally, the number of instructions required in the called routine is 5 instructions on entry and 3 on exit. The call itself also requires 3 instructions.
- The caller saves its link register (the branch-to-register or BASE of the callee) on the stack area and the callee restores it upon return.
- The TPF system does not assume the default register conventions described. The register conventions are used as defaults in the system macros and are encouraged for use by all new or modified CP routines. However, the linkage macros are flexible enough to handle any base and linkage registers that do not conform to the defaults.
- Following are several enhancements that can be made to the linkage described previously:

- Use Save and Restore for only those registers needed by the routine. This enhances performance because fewer registers are saved by a single Load and Store multiple instruction. This technique can be used in performance-oriented routines. For debugging purposes, it is strongly recommended that nonperformance critical routines save and restore all registers.
- Performance-sensitive routines like storage block management routines reference a predefined stack area. That is, R13 will always point to a stack area that contains an area allocated for these routines to eliminate the cost of pushing and popping the stack. This area is overlaid by each call or push of the stack. Therefore, these routines must be the last level of call. This technique is for performance sensitive routines only and will be maintained within the system macros.

Examples of the use of these system macros and the stack area follow.

## Sample Stack Area Use

The following is a sample of the stack area use.

```
(PROG2 SLNKC) R13 →      PERFORMANCE AREA

                          PREV      Pointer to last stk
                          R0        Register
                          R1        save
                          .         area
                          R15

                          USER DEFINED AREA

(PROG1 SLNKC) R13 →      PREV      Pointer to last stk
                          R0        Register
                          R1        save
                          .         area
                          R15

                          USER DEFINED AREA

(PROG0) STACK→ R13→PREV  Pointer to last stk
                          R0
                          R1        Register
                          .         save
                          R15        area

                          USER DEFINED AREA
```

**Note:** Performance area exists only at highest/deepest call. It is overlaid on all other calls.

## Coding Example for Control Program (CP) Linkage

Program 0, which is the initial routine, is the initial entry to the CP environment. Load the stack register with the proper stack from prefix page. Call first routine.

<u>Without Macros</u>	<u>With Macros</u>
.	.
.	.
.	.
L R13,PFSSAVE	L R13,PFSSAVE
.	.
.	.
.	.

```

      .
      ST R15,STKR15(R13)
      L R15,=A(PROG1)
      BASR R14,R15
      .
      .
      .
      LTORG ,
      =A(PROG1)
      .
      .
      .
      CLNKC RTN=PROG1,TYPE=INT
      .
      .
      .
      LTORG ,
      =A(PROG1)

```

End Prog 0 Initial Entry to control program.

Program 1, which is the caller routine, saves registers on the stack. Push the stack pointer. Save the caller's stack pointer.

<u>Without Macros</u>	<u>With Macros</u>
PROG1 DS 0H	PROG1 DS 0H
<pre>           USING P1WK,R13           STM R0,R14,STKR0(R13)           SL R13,=A(P1WKL)           MVC STKPREV(R13),K1WKLEN_STRK13(R13)           P1WK DSECT           STKPRF DS 16X Performance area           STKPREV DS 1F Previous Stack           STKREGS DS 16F Register Save           STKINL DS 6F In-Line Reg Save            P1A DS 1F           P1B DS 1F           P1C DS 1F            DS 0D           P1WKLEN EQU *-P1WK           CSECT           .           .           ST R15,STKR15(R13)           L R15,PROG2A           BASR R14,R15           .           .           .           LM R0,R15,STKREGS           BR R14           .           .           PROG2A DC A(PROG2)           LTORG ,                   =A(P1WKLEN) </pre>	<pre>           SLNKC LOREG=R0,                   DSECT=P1WRK            P1A DS 1F           P1B DS 1F           P1C DS 1F            DLNKC           .           .           .           CLNKC RTN=PROG2,TYPE=INT           .           .           .           RLNKC LOREG=R0           .           .           .           LTORG                   =A(PROG2)                   =A(P1WKLEN) </pre>

End Prog 1

Program 2, which is the called routine, saves registers on the stack. Push the stack pointer. Save the caller's stack pointer.

<u>Without Macros</u>	<u>With Macros</u>
PROG2 DS 0H	PROG2 DS 0H
<pre>           USING P2WK,R13           STM R0,R14,STKR0(R13)           SL R13,=A(P2WKL)           MVC STKPREV(R13),P2WKLEN+STKR13(R13) </pre>	<pre>           SLNKC LOREG=R0,                   DSECT=P2WK </pre>



- Entry control block (ECB) equates
- Communications Line equates
- System equates
- Configuration-dependent system equates
- Tape program equates
- Terminal type equates
- Control program equates.

In addition to system macros that consist purely of equates, DSECTS also often carry equates with them.



---

## System Macros

This chapter contains an alphabetic listing of the TPF system macros. The description of each macro includes the following information:

**Format:** Provides a syntax (railroad track) diagram for the system macro and a description of each parameter and variable. See “How to Read the Syntax Diagrams” on page xv for more information about syntax diagrams.

**Entry Requirements:** Lists any special conditions that must be true on entry to the system macro.

**Return Conditions:** Lists what is returned when the system macro finishes processing.

**Storage Requirements:** Provides the number of bytes (or range of bytes) required by the system macro.

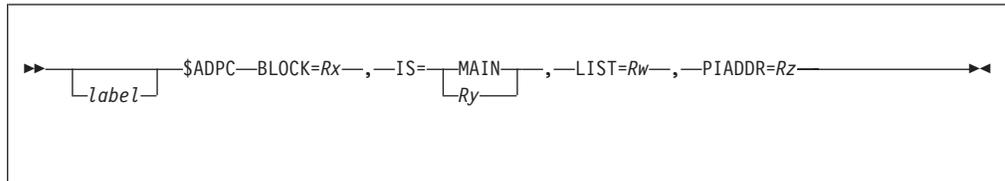
**Programming Considerations:** Lists any additional considerations for using the system macro, including any restrictions or limitations.

**Examples:** Provides one or more examples that show you how to code the system macro.

## \$ADPC—Add Work to a List on Specified I-Stream

Use this system macro to add a specified data block to a specified dispatch list on a specified I-stream. A work element can be added to any I-stream from any I-stream.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rx**

The specified register is set to the address of the data block to be added to the list.

**IS** Specify one of the following:

#### **MAIN**

The main I-stream is the target.

#### **Ry**

The specified register is set to the target I-stream number.

#### **PIADDR=Rz**

The register specified is set to the address of the routine to be activated when the data block is dispatched from the list on the target I-stream.

#### **LIST=Rw**

The specified register is set to the appropriate dispatch list index value as defined in the CLHEQ macro (Ready List, Input List, Defer List).

### Entry Requirements

The PSW protect key must be 0 and must be in supervisor state.

### Return Conditions

- Control is returned to the next sequential instruction. The contents of all registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- An item cannot be added to the Cross List with this macro.
- Return is made to the next sequential instruction.
- This macro generates a CLNKC to the service routine.
- For entry to the macro service routine, the macro will convert the input to the following:

R0 PIADDR, postinterrupt address routine

R1 BLOCK address

R2 LIST index as defined in CLHEQ

R3 I-stream number (in the range 1–16).

- CAPT must be called by the CSECT.
- \$ADPC will bypass normal dispatch management shutdown processing when adding to any list other than the main I-stream input list.
- Depending on the *from* I-stream and the *to* I-stream, the \$ADPC service routine may use the \$CRISC macro to cross between I-streams.
- This macro is for use in the control program (CP) only.
- An entry for this macro will be added to the macro trace table.
- This macro requires that if the address passed as the BLOCK parameter is an ECB address, it must be the SVA address for the ECB. IOB and SWB block addresses are the same in SVA and EVA. SVA addresses are required because the processing routine runs in the SVM.

You must understand address space considerations when coding this macro.

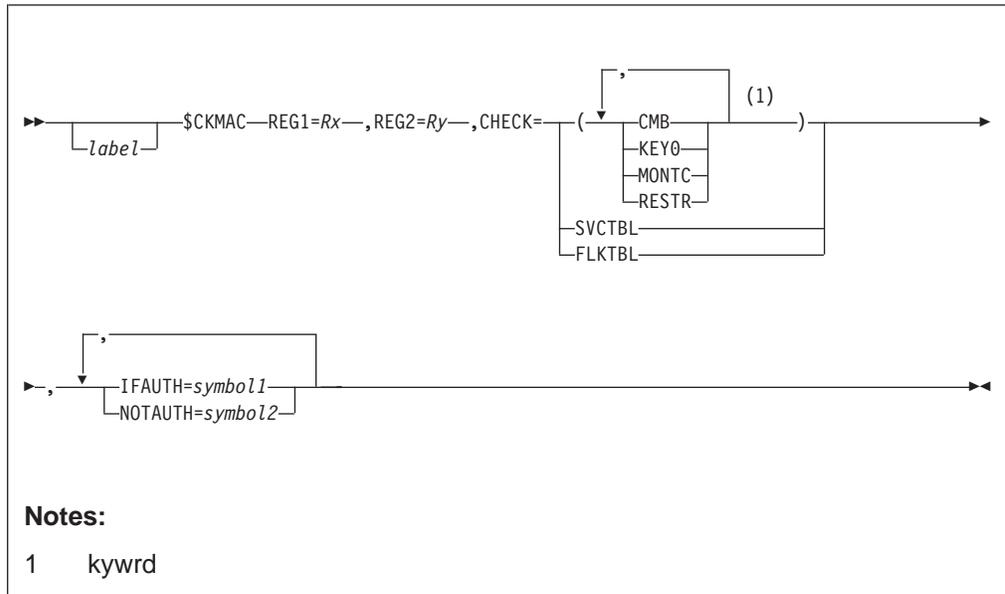
## Examples

None.

## \$CKMAC—Validate Use of Restricted Macro

Use this system macro to validate or reject authorization of E-type programs requesting to run a restricted use macro. Macro service routines identified as restricted use it to enforce the restriction. The program allocation table (PAT) entry of the requesting program is checked to determine whether the requesting program is authorized to process a class of restricted use macros.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### CHECK

This required parameter specifies the class authorization to be verified. There are three possible operands:

#### (kywrd)

Specify one of the following:

#### KEY0

The KEY0 option tests for storage protect key 0 authorization.

This option is listed as Key0 in the authorization box.

#### MONTC

The MONTC option tests for supervisor state authorization.

This option is listed as System in the authorization box.

#### RESTR

The RESTR option tests for other restricted macro use authorization.

This option is listed as Restricted in the authorization box.

#### CMB

The CMB option tests for authorization to obtain common storage blocks.

This option is listed as Common Storage in the authorization box.

**SVCTBL**

The SVCTBL option tests the SVC table entry for the macro against the PAT entry for the program.

**FLKTBL**

The FLKTBL option tests the fast link table entry for the macro against the PAT entry for the program.

**REG1=Rx**

A work register must be specified. R0 cannot be used.

**REG2=Ry**

A work register must be specified. R0 cannot be used.

**IFAUTH=*symbol1***

If this parameter is coded, and authorization is validated, control will be passed to the location defined by *symbol1*.

**NOTAUTH=*symbol2***

If this parameter is coded, and authorization is rejected, control will be passed to the location defined by *symbol2*.

**Note:** Either IFAUTH or NOTAUTH must be coded. Both can be coded.

## Entry Requirements

- R0 cannot be specified as a work register.
- R9 must contain the address of the requesting ECB.

## Return Conditions

- Control is returned to the location specified by the appropriate authorization check condition or to the next sequential instruction (NSI).
- The register specified by the REG parameter has been used as a work register. Its contents are unpredictable.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is used by the macro decoder and by restricted use macro service routines to validate a requestor's (E-type program) authorization to execute a macro.
- This macro generates inline code.
- Authorization is validated through ISV0SV field ISV0AUTHZ.
- If coding a SNAPC macro to enforce an authorization failure, be sure to include the SVC Old PSW (if an SVC), by using the SNAPC LIST parameter and the registers, by using the SNAPC REGS=YES parameter. This will provide consistency in \$CKMAC usage.
- Some macros have authorizations required to use particular parameters even though the macros themselves do not require authorization by the CKMAC parameter. The restricted parameters are documented in their respective macro descriptions.

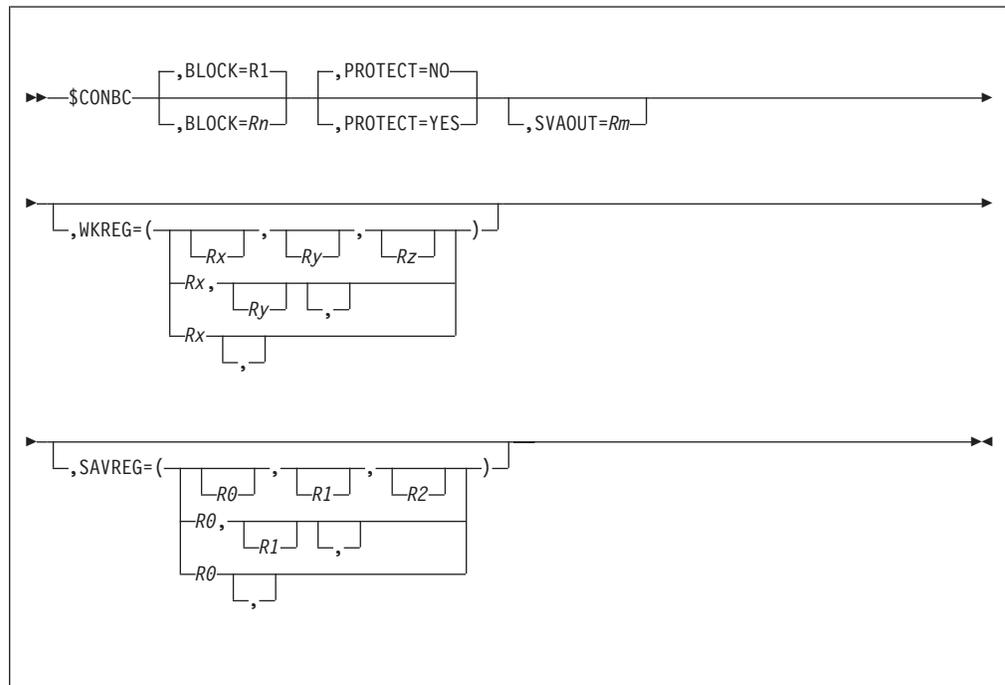
## Examples

None.

## \$CONBC—Connect Block to ECB Virtual Memory

Use this system macro to attach a data block to an entry control block (ECB) address space.

### Format



#### BLOCK=R1|Rn

Use this parameter to specify the system virtual address (SVA) of a storage block. The register specified contains the SVA address of a storage block. The block must be of a type that can be attached to ECB CBRWs (128, 381, 1055, or 4095 bytes).

On return, this register contains the ECB virtual address (EVA) of the connected block.

R9 and R13 cannot be used with the BLOCK parameter R9 cannot be used because you may not attach the ECB to itself. R13 cannot be used because you may not attach the stack to the ECB.

R1 is the default.

#### PROTECT

Use this parameter to specify if the ECB is allowed to modify data in the block.

#### YES

If YES is specified and the block resides in main storage above the frames, the page protection bit is set in the page table entry for the storage, and the data is protected against modification.

**Note:** The YES option of the PROTECT parameter is intended exclusively for the ENTxC macros to connect a 24-bit program residing in a 4 K VFA block to an ECB. The YES option is ineffective when the block resides in or below the frames.

**NO**

If NO is specified or if the block resides in main storage in or below the frames, the page protection bit is not set in the page table entry and the data can be modified. NO is the default.

**SAVOUT=Rm**

This optional parameter specifies a register that contains the SVA of the block connected by the macro.

**Note:** The SVA may change during the process of connecting the block to the ECB.

The BLOCK and SVAOUT parameters should not specify the same register. There is no default register for SVAOUT.

**SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

**WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- R9 should not be specified for the WKREG parameter.
- R9 must point to the ECB to which the block will be connected.
- You can run the \$CONBC macro from system virtual memory (SVM) or ECB virtual memory (EVM).

**Notes:**

1. Keep in mind that you can run this macro from EVM only if the block being connected is addressable by the ECB virtual address (EVA).
  2. When blocks are being attached in frames, the program running this macro must be in SVM.
- The block pointed to by the BLOCK parameter must be disconnected from the ECB by issuing \$GETBC ECB=NO or \$DISBC.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register specified in the BLOCK parameter contains the EVA address of the connected block.
- The condition code is not preserved.

## **\$CONBC**

### **Programming Considerations**

- This macro must be processed on the same I-stream as the ECB to which the block will be connected.
- This macro is for use in the control program (CP) only.
- System error dumps can occur when servicing a \$CONBC request. See *Messages (System Error and Offline)* for more information.

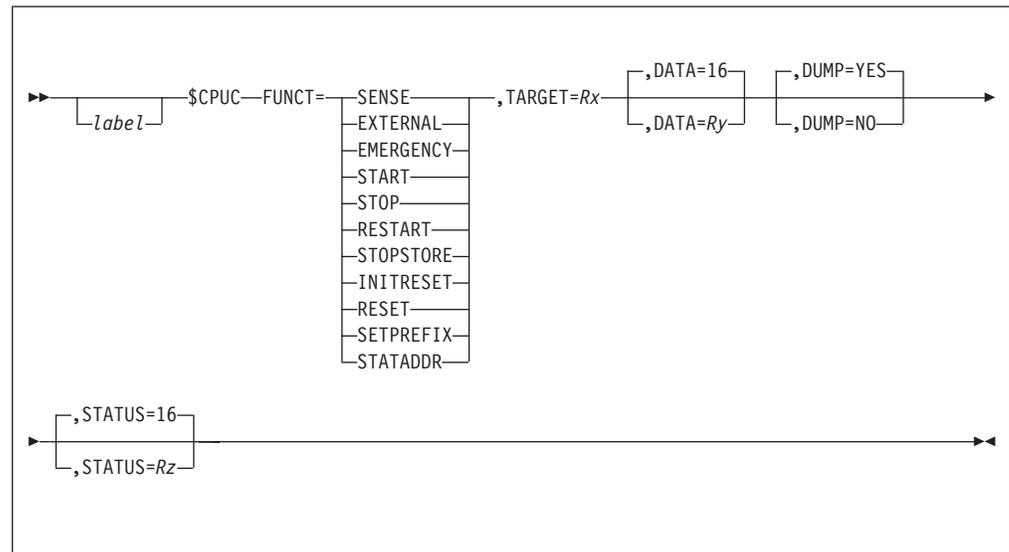
### **Examples**

None.

## \$CPUC—Interface for SIGP Services

Use this system macro to provide an interface to the CCIISC inter-I-stream control CSECT for SIGP services. (Implemented functions are strictly those functions provided by the SIGP instruction.)

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **DATA**

Specify one of the following:

**Ry** The specified register is set to the address of a data area for SIGP to store into.

**16** This is the default. There will be no data area.

#### **DUMP**

Specify one of the following:

##### **YES**

This is the default. A system error is taken on a SIGP error.

##### **NO**

No dump is taken.

#### **FUNCT=x**

Where x is one of the following SIGP functions:

##### **SENSE**

(01)

##### **EXTERNAL**

(02)

##### **EMERGENCY**

(03)

##### **START**

(04)

## \$CPUC

**STOP**  
(05)

**RESTART**  
(06)

**STOPSTORE**  
(09)

**INITRESET**  
(0B)

**RESET**  
(0C)

**SETPREFIX**  
(0D)

**STATADDR**  
(0E)

### STATUS

Specify one of the following:

*Rz* The register specified is available for output status.

**16** This is the default. There is no output status.

### TARGET=*Rx*

The specified register is set to the CPU address of the target CPU that FUNCT will be run against.

## Entry Requirements

The PSW protect key must be 0 and in supervisor state.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- General registers R14 and R15 are used to link to the service and cannot be used as parameters.
- Return is to the NSI, with the DATA area and STATUS register set to indicate the result of the requested function. All other registers will be restored.
- The SIGP STATUS register values are:

CC0 Return	X'00000000'	Order Code accepted
CC3 Return	X'FFFFFFFF'	CPU NOT Operational
CC2 Return	X'FFFFFFFF'	CPU Busy
CC1 Return...		Stored Status
SIGCHECK...	X'80000000'	EQUIPMENT CHECK
SIGISTATE...	X'00000200'	INCORRECT STATE
SIGIPARAM...	X'00000100'	INCORRECT PARAMETER
SIGEXCALL...	X'00000080'	EXTERNAL CALL PENDING
SIGSTOPPD...	X'00000040'	STOPPED
SIGOPINTR...	X'00000020'	OPERATOR INTERVENING
SIGCHKSTP...	X'00000010'	CHECK STOP

```
SIGSPINOP... X'00000004' SERV PROC INOPERATIVE  
SIGINORDR... X'00000002' INVALID ORDER  
SIGRCVCHK... X'00000001' RECEIVER CHECK
```

- This macro can be used in a special package only.

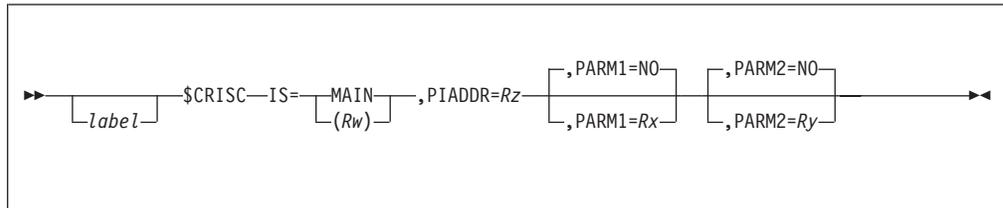
## **Examples**

None.

## \$CRISC—Cross Over to Another I-Stream

Use this system macro to add the specified item to the CROSS list in the specified I-stream. An element of work cross over to the main I-stream from any I-stream or from the main I-stream to any other I-stream.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**IS** Specify one of the following:

**MAIN**

The main I-stream is the target I-stream.

*Rw*

The specified register is set to the target I-stream number.

**PARM1=*Rx*|NO**

The specified register contains user parameter 1. The default is NO.

**PARM2=*Ry*|NO**

The specified register contains user parameter 2. The default is NO.

**PIADDR=*Rz***

The register specified is set to the address of the routine to be activated when the data block is dispatched from the cross list on the target I-stream.

### Entry Requirements

- The PSW protect key must be 0 and must be in supervisor state.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- For entry to the macro service routine, the macro will convert the input to the following:
  - R0 PARM1, which is a 4-byte parameter 1
  - R1 PARM2, which is a 4-byte parameter 2
  - R2 PIADDR, which is a postinterrupt routine address
  - R3 I-stream number (in the range 1–16).
- CAPT must be called by the CSECT.

## **\$CRISC**

- You must use the \$CRISC macro to change I-streams. However, the \$CRISC macro does **not** update I-stream related fields in the ECB or low main storage (for example, CA3NEBA).
- This macro is for use in the control program (CP) only.
- An entry for this macro will be added to the macro trace table for both this I-stream and the target I-stream.

## **Examples**

None.

## \$DCOLC

### \$DCOLC–Data Collection Hook Insertion

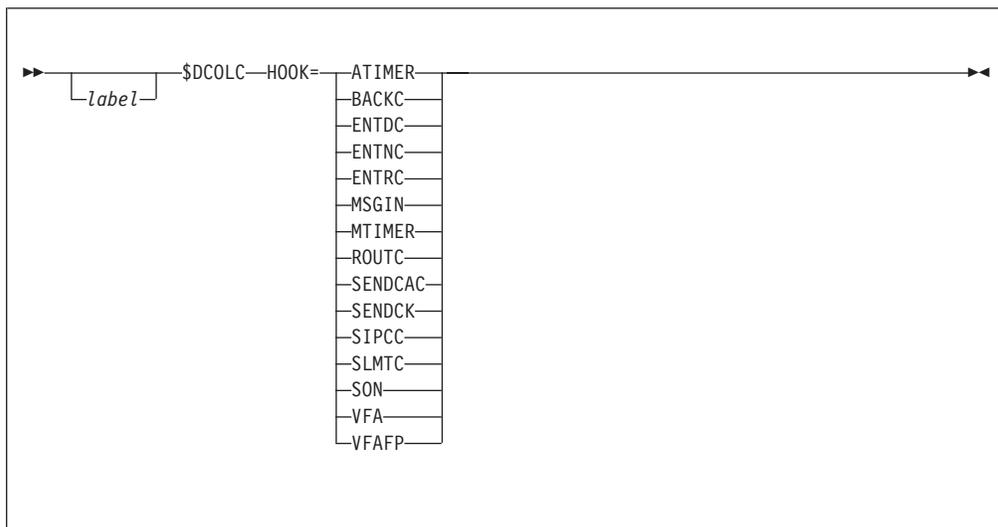
This system macro is for use by TPF system data collection only. It must not be used and calls to it must not be modified by anyone. This system macro provides an interface for data collection information. Information can be collected for:

- Message traffic (ROUTC, SENDC, SIPCC, MSGIN, SLMTC)
- File storage activity handled by SON and VFA.

The \$DCOLC macro generates either an SVC linkage to a hook (in E-type code) or the hook itself (in C-type code). When active, the hook calls the corresponding collection routine.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **HOOK**

A required parameter specifying the intercept type of the inline code to be generated:

##### **ATIMER**

Intercepts CPU timer external interrupts on the application I-streams.

##### **BACKC**

Intercepts BACKC macros

##### **ENTDC**

Intercepts ENTDC macros

##### **ENTNC**

Intercepts ENTNC macros

##### **ENTRC**

Intercepts ENTRC macros

**MSGIN**

Intercepts incoming messages

**MTIMER**

Intercepts CPU timer external interrupts on the main I-streams.

**ROUTC**

Intercepts ROUTC macros

**SENDCAC**

Intercepts SENDC A/C macros

**SENDCK**

Intercepts SENDC K macros

**SIPCC**

Intercepts SIPCC macros

**SLMTC**

Intercepts SLMTC macros

**SON**

Intercepts find/file macros handled by SON

**VFA**

Intercepts find/file macros handled by VFA

**VFAFP**

Intercepts find/file macros handled by VFA fast path

## Entry Requirements

The entry requirements for each HOOK type are:

Hook	Register(s)	Register Description	Mode	Restriction
BACKC	R9	Address of ECB	EVM	CP only
ENTDC	R9	Address of ECB	EVM	CP only
ENTNC	R9	Address of ECB	Note	CP only
ENTRC	R9	Address of ECB	EVM	CP only
MSGIN	R9	Address of ECB	EVM	none
ROUTC	R9	Address of ECB	EVM	CP only
SENDCAC	R9	Address of ECB	EVM	CP only
SENDCK	R9	Address of ECB	EVM	CP only
SIPCC	R9	Address of ECB	EVM	CP only
SLMTC	R9	Address of ECB	EVM	CP only
SON	R1	Address of MIO	EVM	CP only
	R9	Address of ECB		
VFA	R1	Address of MIO	EVM	CP only
	R9	Address of ECB		
VFAFP	R1	Address of MIO	EVM	CP only
	R9	Address of ECB		

## **\$DCOLC**

**Note:** In the case of the ENTNC hook, when CE3PAT is not equal to 0, the addressing mode is EVM. When CE3PAT equals 0, the addressing mode is SVM.

### **Return Conditions**

- If the intercepted entry corresponds to a collector that is currently active, data for that entry is conditionally collected and the counter for skipping intercepts is changed.
- No user fields change when this macro processes.

### **Programming Considerations**

This macro is restricted to the data collection package only and should not be coded by any other user. The macro is called throughout the TPF system. It is used to provide an interface to data collection.

### **Examples**

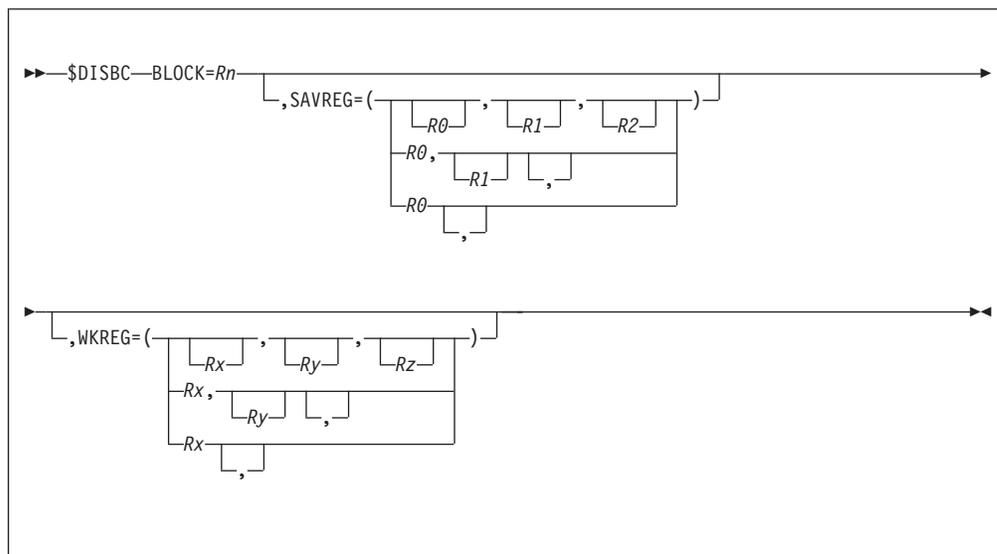
The following example generates the inline code for data collection to conditionally intercept the ENTDC macro call.

```
$DCOLC HOOK=ENTDC
```

## \$DISBC—Disconnect A Block from the ECB Virtual Memory

Use this system macro to disconnect a storage block from the entry control block (ECB) virtual memory (EVM). The macro is used by the TPF system for data that should no longer be accessible to the ECB (such as output messages after the ROUTC macro). The TPF system also uses this macro to support macros (such as the CREEC macro) that pass main storage blocks from one ECB to another.

### Format



#### BLOCK=Rn

This required parameter specifies the register containing the 31-bit EVM address of the block to be disconnected. R13 can not be specified for this register. The block must be of a type that can be attached to ECB core block reference words (CBRWs) (128, 381, 1055, or 4095 bytes).

On return, the register contains the 31-bit SVM address of the disconnected block.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## \$DISBC

### Entry Requirements

- This macro can be run in the EVM or the SVM.
- R9 must point to the ECB defining the EVM in whatever address space this macro is called.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The condition code is not preserved.

### Programming Considerations

- This macro must be processed on the same I-stream as the ECB which provides the address space for the EVA address.
- This macro is for use in the control program (CP) only.
- The SVM address of the disconnected block may **not** be the same as the SVM address of the block before \$DISBC.
- If the block contains imbedded pointers (for example, the first fullword in the block points to the start of a parameter list in the block), then once \$DISBC has been issued, these pointers must be relocated to be useful.

For example, before the \$DISBC, the EVM block address might be X'10000' and the SVM block address might be X'1FE3000'. After \$DISBC, the SVM block address might be X'2301000' and the block does not have an EVM address. Therefore, in this example, an imbedded pointer of X'100008' (EVM address) or X'1FE3008' (SVM address) would need relocation to an SVM address of X'2301008'.

- System error dumps can occur when servicing a \$DISBC request. See *Messages (System Error and Offline)* for more information.

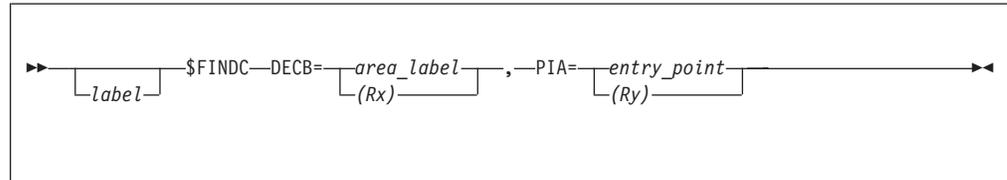
### Examples

None.

## \$FINDC-CP Find A File Record

Use this system macro to read a record from a file into storage. The record can be in the virtual file access (VFA) area.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **DECB=area\_label(*Rx*)**

The address of a data event control block (DECB) must be specified either by any register (other than R0) or by the label. The label is the address of the DECB, not an address to a pointer of the DECB.

#### **PIA=entry\_point(*Ry*)**

The address of an entry point to be given control when the request has completed. The address can be specified either by a register or by the name of the entry point if it is internal to this routine. If the entry point is external to this routine, a register must be used.

### Entry Requirements

- The DECB address must have the following fields set in the IDECB DSECT:
 

IDECECB	0
IDECDBI	Set to the database ID (DBI) of the subsystem
IDECSSU	Set to the correct subsystem user (SSU) for the file address
IDECDAD	Address of the storage area to hold the record
IDECDLH	Length of the storage area
IDECRID	Record ID of the record
IDECRCC	Record code check (RCC) value
IDECFA	File address for the record.
- The TPF system state must be:
 

Address Mode	31-bit
Address State	SVM
PSW KEY	0
PSW MASK	I/O disabled
System State	Privileged.
- Register 0 cannot be used to pass the DECB address.

### Return Conditions

- \$FINDC return:
  - Control is returned to the next sequential instruction.

## \$FINDC

- R14 contains the address of the next sequential instruction.
- The contents of R0 and R1 are destroyed across this macro call. The contents of all other registers are preserved.
- PIA entry point
  - Register contents on entry:
    - R1     DECB address
    - R2     PIA entry point address
    - R14    Return address (CPU loop).
  - The TPF system state on entry:
    - Address Mode   31-bit
    - Address State   SVM
    - PSW KEY        0
    - PSW MASK       I/O Disabled
    - System State   Privileged.
  - In the post-interrupt routine, control is returned on the same I-stream in which the \$FINDC macro was run.
  - The IDECSUD field shows the success or failure of the request.

Error codes	Descriptions
X'00'	I/O completed without errors (block returned)
X'40'	DASD (hardware) I/O error (block not returned)
X'80'	Record ID error or record code check error (block returned)
- If the macro returns normally, the storage block contains the specified record. If there is an End-of-File condition or a software or hardware error, the contents of the specified storage block cannot be predicted.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is for use in the control program (CP) only.
- The control program (CP) caller must do several things:
  1. Allocate a DECB and fill in the required values.
  2. Allocate a storage block to hold the specified record and store the storage block's address in the DECB.
  3. Issue a \$FINDC macro to cause the DASD support to read the specified record from file into the specified storage area.

On return from the macro, the requested operation has only been accepted. When the request has completed, the entry point specified in the PIA parameter is given control and the success or failure of the request is indicated in the DECB.
  4. In the post-interrupt routine, control is returned in the same I-stream on which the \$FINDC macro was run.
- Various error conditions can occur:
  - A check is made by the control program (CP) to determine whether the DECB is now active. If the DECB is active, control is transferred to the system error routine.

## **\$FINDC**

- If the file address contained in the DECB is not valid, an error code (X'02') is indicated in the DECB.
- The control program (CP) verifies that the specified record ID matches the ID in the record. If the record ID specified is zero, this check is not made. If the match fails, an error code is indicated in the DECB.
- The specified record code check is verified with the record code check in the record. This check is not made if the specified record code check is zero. If the check fails, an error code is indicated in the DECB.

## **Examples**

None.

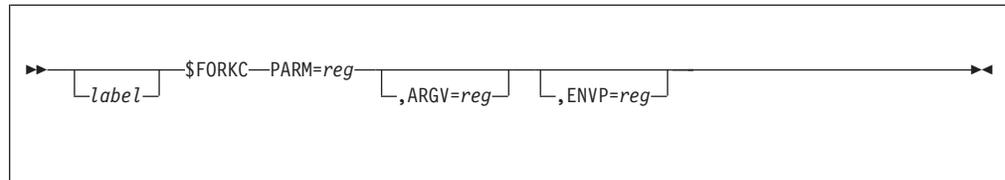
## \$FORKC

### \$FORKC—Create an Asynchronous ECB

Use this system macro to create an asynchronous entry control block (ECB) on a specified I-stream.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **PARM=reg**

Specifies the macro parameter list that is mapped by the IFORK DSECT, where *reg* is a register from R0 to R7.

#### **ARGV=reg**

Specifies the macro parameter list that is mapped by the IDARGV DSECT, where *reg* is a register from R0 to R7.

#### **ENVP=reg**

Specifies the macro environment variable list that is mapped by the IDENV DSECT, where *reg* is a register from R0 to R7.

### Entry Requirements

- This macro is restricted to ECB-controlled programs.
- This macro is intended only for `tpf_fork` library routine use. The interface for this macro is subject to change.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The process ID of the created child process is stored in the parameter list.

### Programming Considerations

- See the `tpf_fork` function in the *TPF C/C++ Language Support User's Guide* for information about the items that the child ECB inherits from the parent ECB.
- The macro trace information for this macro will contain the program that will be activated and the I-stream on which the program is running.

### Examples

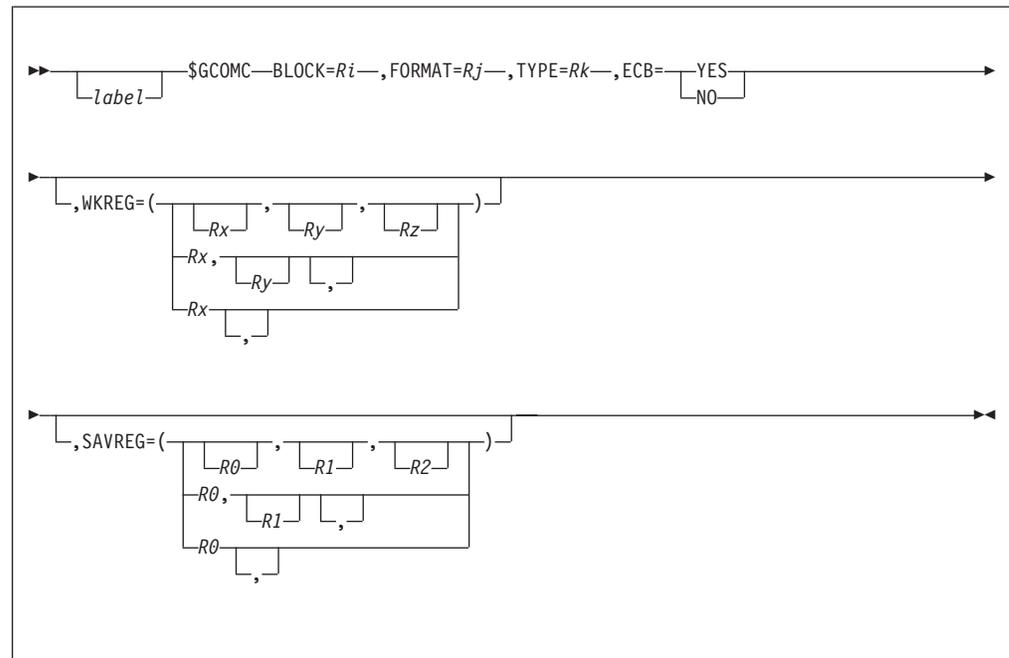
The following example creates an asynchronous ECB.

```
$FORKC PARM=R5
```

## \$GCOMC—Obtain Common Storage Block

Use this system macro to obtain a storage block from the pool of working storage that is accessed at the same address in any address space.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Ri**

This parameter is used to return the block address. The register specified is loaded with the storage block address on return from the service routine. The block is allocated below 16 MB. The block address is correct in 24-or 31-bit mode and in any address space. Any general purpose register, with the exception of stack register R13, can be used for this parameter.

#### **FORMAT=Rj**

This parameter is used to specify the format flag to be placed in the block. The register specified contains the block format flag in the rightmost (low-order) byte. Any general purpose register, with the exception of stack register R13, can be used for this parameter.

#### **TYPE=Rk**

This parameter specifies the storage block type requested. The register specified in this input parameter contains the logical storage block type equate value in the low-order (rightmost) byte. The block equate is restricted to blocks that can be placed on an ECB core block reference word (L0, L1, L2, or L4). Any general purpose register, with the exception of stack register R13, can be used for this parameter.

#### **ECB**

This parameter indicates which address will be saved in the common block control table.

## \$GCOMC

### YES

Indicates the SVA of the ECB (CE2SVA) is to be saved in the CCTECB field of the common control tree table (IDSCCT)

### NO

Indicates that the address of the caller (in R14) should be saved in the CCTECB field.

### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- Entry can be in either EVM or SVM addressing mode.
- The program calling this macro must be running in privileged mode with a zero protection key storage.
- R13 is assumed to contain the stack pointer.
- R9 is assumed to contain the ECB pointer.

## Return Conditions

- Control is returned to the NSI.
- The register specified in the BLOCK parameter contains the storage block address.
- The storage block returned has the format flag set to the value specified by the FORMAT parameter.
- The register specified in the FORMAT parameter contains the logical block type and the logical size of the storage block in a format suitable for storing in a core block reference word.
- The condition code is not saved.

## Programming Considerations

The most efficient use of this macro can be accomplished by using the WKREG parameter with the SAVREG parameter.

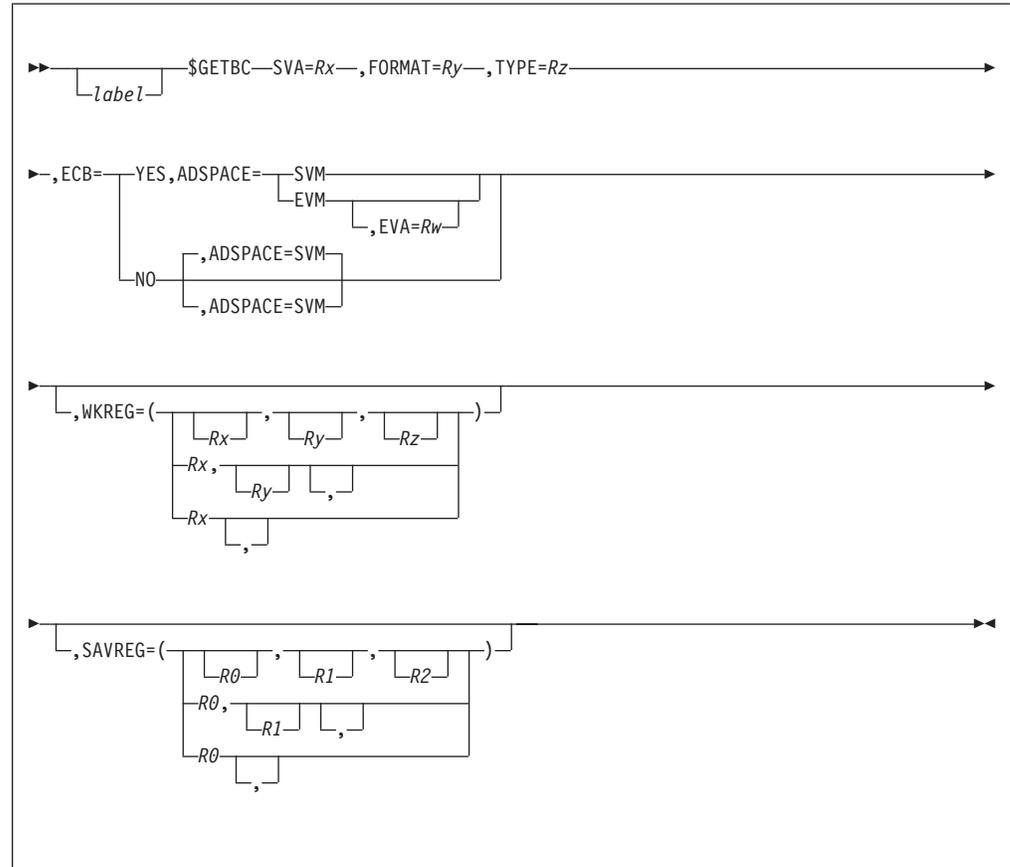
## Examples

None.

## \$GETBC—Obtain Storage Block

Use this system macro to obtain a storage block from the control lists and connect that block to the entry control block (ECB) virtual memory (EVM).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **SVA=Rx**

The register in which the SVM block address is returned. The register specified is loaded with the SVM storage block address on return from the service routine. All registers, except register R13, are valid. The SVA parameter is always valid. It is optional, except when the EVA parameter is invalid or omitted.

#### **FORMAT=Ry**

This parameter specifies the format of the block. The register specified contains the block format flag in the rightmost (low-order) byte. All registers, except register R13, are valid. No checking is done to ensure the validity of the format flag supplied.

#### **TYPE=Rz**

This parameter specifies the storage block type requested in the low-order (rightmost) byte. The register specified contains the logical storage block type equate (L0, L1, L2, or L4). All registers, except register R13, are valid.

## \$GETBC

### ECB

This parameter specifies whether the storage acquired is connected to the ECB virtual memory (EVM) immediately.

#### YES

When ECB=YES, the frame containing the block is connected, thereby becoming part of the ECB's virtual memory.

#### NO

When ECB=NO, the frame containing the block is not connected to an ECB's virtual memory (EVM) at this time. It can later be connected via the \$CONBC macro. ADSPACE=EVM may not be coded and ADSPACE=SVM is the default. The EVA parameter may not be coded.

### ADSPACE

This parameter gives the address space where the calling code is operating.

#### EVM

The ECB virtual memory (EVM) is the address space used by ECB-controlled programs. There is one for each ECB in the TPF system. Control program (CP) code can also be executing in the EVM on behalf of the ECB.

#### SVM

The system virtual memory (SVM) is used mainly by control program (CP) post interrupt routines and interrupt handlers. It is never used by ECB-controlled programs.

### EVA=R<sub>w</sub>

The register in which the EVM block address is returned. The register specified is loaded with the EVM storage block address on return from the service routine. All registers, except register R13, are valid. The EVA parameter is valid only when ECB=YES is coded. When the SVA parameter is omitted, the EVA parameter is mandatory.

### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- The program calling this macro must be running in privileged mode with a zero protect key storage. The program must also be operating in 31-bit mode. The address space of the code issuing this macro must be correctly specified via the ADSPACE parameter.
- R13 is assumed to contain the stack pointer.

- If the ECB=YES parameter is used, R9 is assumed to point to the ECB and the ECB itself is assumed to be running on the current I-stream.

## **Return Conditions**

- Control is returned to the NSI.
- The register specified in the EVA parameter contains the EVM storage block address.
- The register specified in the SVA parameter contains the SVA storage block address.
- The register specified in the FORMAT parameter contains the logical block type and the logical size of the storage block in a format suitable for storing in a core block reference word.
- The storage block returned has the format flag set to the value specified by the FORMAT parameter.
- The condition code is not saved.

## **Programming Considerations**

This macro is for use in the control program (CP) only.

## **Examples**

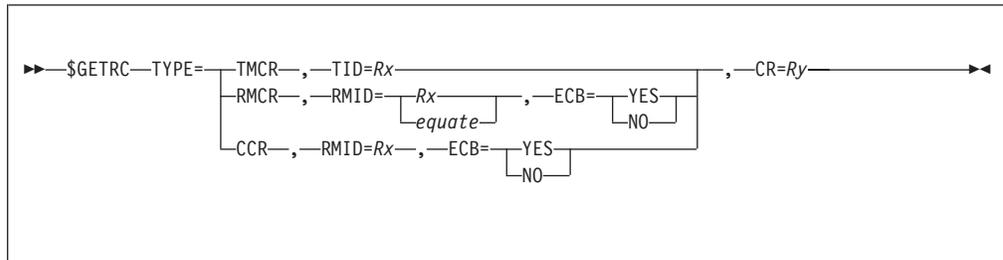
None.

## \$GETRC

### \$GETRC – Get a Control Record

Use this system macro to get one of the control records associated with TPF transaction services processing. The \$GETRC macro is only for use by the transaction manager (TM) and resource managers (RMs). Through the use of the \$GETRC macro, the TM and RMs are able to define and control the commit scope environment.

### Format



#### TYPE

Defines the type of control record requested. This parameter is required. The following control record types are valid:

#### TMCR

Requests a transaction manager control record (TMCR). A new TMCR is retrieved and chained to the previous one. The previous TMCR is pushed.

#### RMCR

Requests a resource manager control record (RMCR). This parameter returns a pointer to an RMCR associated with the current TMCR. It does not retrieve a new RMCR.

#### CCR

Requests a commit control record (CCR). A new CCR is retrieved and chained to the previous one.

#### TID=Rx

A register (R1 through R7) containing a pointer to the transaction ID (TID) to be associated with this request. This parameter is required when TMCR is coded.

#### CR=Ry

A register (R1 through R7) to contain the address of the retrieved control record. This parameter is required.

#### RMID=Rx

A register (R0 through R7) containing the resource manager ID (RMID) associated with the record request. This parameter is required when the RMCR or CCR record type is coded.

**Note:** An equate can be specified when you specify TYPE=RMCR.

#### ECB

Defines the area to which the TMCR is anchored. This parameter is required when the RMCR or CCR record type is coded.

#### YES

Specifies that the TMCR is anchored out of the ECB at the CE2TMCR field and out of the transaction anchor table (TANC).

**NO**

Specifies that the ECB TMCR anchor is not to be used. Instead, the TMCR address is supplied in the register specified in the CR parameter. The TMCR is still anchored to the TANC.

**Entry Requirements**

- When you specify ECB=YES, R9 must contain the address of the ECB being processed. Control records are anchored off of the ECB through the CE2TMCR field. The \$GETRC macro uses and updates this field.
- When you specify ECB=NO, the register, which is specified by the CR parameter, must contain the address of the current TMCR.

**Return Conditions**

- Control is returned to the next sequential instruction.
- The contents of the registers are preserved across this macro call with the exception of the registers specified by the macro parameters. The contents of these registers are as follows:
  - The register specified on the CR parameter will contain the address of the requested record.
  - The register specified on the RMID or TID parameters remains unchanged.
- When you specify the TMCR record type, the TMCR control fields are set by the macro and all other fields are cleared. Each RMCR entry is also cleared. If this is a root TMCR, it is also anchored off the TANC. Additionally, ECB field CE2TMCR is updated to point to the new TMCR.
- The condition code (CC) is not preserved across the macro call.

**Programming Considerations**

- This macro can be run from any I-stream.
- This macro can only be called from control program (CP) code.
- When getting an RMCR, the RM is responsible for initializing the following fields defined in IRMCR(ICRCR):
  - IRM\_XID
  - IRM\_STA
  - IRM\_SIZE.

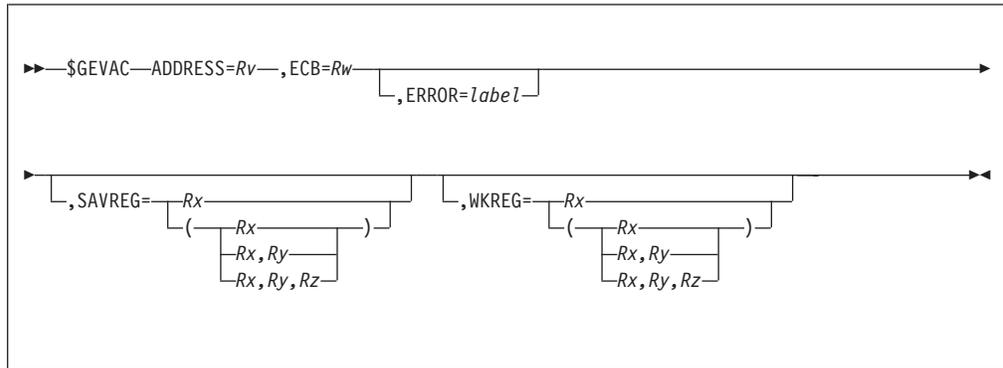
**Examples**

```
$GETRC TYPE=TMCR,CR=R6,TID=R2
$GETRC TYPE=RMCR,CR=R1,RMID=R6,ECB=YES
.
$GETRC TYPE=CCR,CR=R1,RMID=R6,ECB=YES
.
.
```

## \$GEVAC—Convert SVM Address to EVM Address

Use this system macro to convert an address valid in the system virtual memory (SVM) into the corresponding address in the entry control block (ECB) virtual memory (EVM).

### Format



#### ADDRESS=Rv

This required parameter specifies a register containing a 31-bit system virtual memory (SVM) address.

On return, the specified register contains the 31-bit ECB virtual memory (EVM) address corresponding to the input 31-bit SVM address, if one exists. If the SVM address is invalid or does not have a corresponding EVM address, then the high order bit of the input SVM address is turned on.

#### ECB=Rw

This parameter specifies the 31-bit address of the ECB. The 31-bit ECB virtual memory address returned by the \$GEVAC macro is valid only for a single ECB. The specified register is not altered.

**Note:** The ECB address, specified in the ECB parameter, must be in the same addressing mode (SVM or EVM) as the address of the calling routine.

#### ERROR=label

This optional parameter specifies a label identifying the error routine to receive control if an error condition is raised. An error will occur when the EVM address specified in the ADDRESS parameter is invalid.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the

number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- This macro can be called from control program (CP) code only, be run in either EVM or SVM, and be run on any I-stream.
- The ECB specified by the ECB parameter must be operating on the I-stream where the \$GEVAC macro was issued.

## Return Conditions

- Following a successful address conversion, control is returned to the next sequential instruction (NSI).
- If the address can not be converted then the ADDRESS parameter will contain the input address with the high order bit set on to indicate a conversion error. On errors, control is returned to the NSI, except when the ERROR parameter designates an error routine.
- The condition code is not preserved.

## Programming Considerations

The register specified for the ADDRESS parameter cannot be the same register that is specified for the ECB parameter.

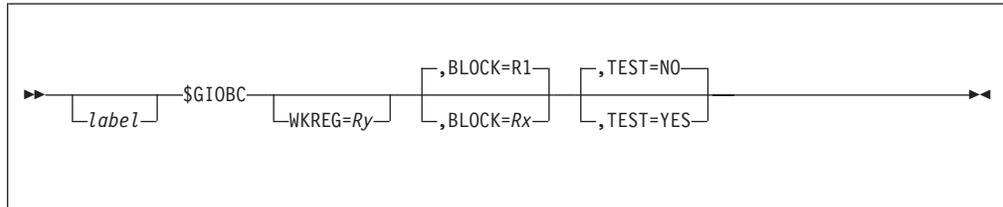
## Examples

None.

## \$GIOBC—Get Available I/O Control Block Address

Use this system macro to obtain an available input/output block (IOB) storage block address.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=R1|R<sub>x</sub>**

This optional parameter specifies the register that will contain the address of the block that was acquired. The default is R1. R0 can not be used.

#### **WKREG=R<sub>y</sub>**

This optional parameter is used to define a work register to be used by the macro service. There is no default. R0 can not be used.

#### **TEST**

This parameter is used for testing and problem determination purposes.

#### **NO**

It is intended that this is used for production systems. The code to get IOBs expands in-line.

#### **YES**

This is used in a test environment when you want to generate code that will go to a central routine (see the CL\$GIOBC entry point in CLHV in the CCSTOR CSECT) to get an IOB. It is useful in monitoring IOB list accesses. It only takes effect if the global variable SBCTEST is set on (B'1') within this routine.

The default is NO.

### Entry Requirements

- Use this macro in the control program (CP) only.
- Programs invoking this macro must be running with a storage protection key of zero, and be in 31-bit addressing mode.
- Programs can be processing in the entry control block (ECB) virtual memory (EVM) or system virtual memory (SVM) address space.

### Return Conditions

- Control is returned to the NSI.
- The TPF system is ended with a catastrophic error if there are no IOBs available.
- When the TPF system is out of IOBs, the request fails and a system error is issued.

- The register specified in the BLOCK parameter contains the storage block address. If the BLOCK parameter was omitted, the storage block address will be in R1 on output.

## Programming Considerations

- This macro can be run on any I-stream.
- System work blocks should replace I/O control blocks for all system functions. Exceptions should be made only for extremely performance critical applications, such as DASD I/O operations.
- The address returned in the register specified in the BLOCK parameter is a 31-bit address, and may point to an IOB allocated above the 16 megabyte boundary. Code must be running in 31-bit mode to access data above the 31-bit boundary.
- The condition code is not set to any specific value on a successful block allocation, and should not be tested.
- System error dumps can occur when servicing a \$GIOBC request. See *Messages (System Error and Offline)* for more information.
- If the TEST parameter is used in an online system, severe system degradation results due to I/O constraints.
- The TEST parameter must only be used in test environments.

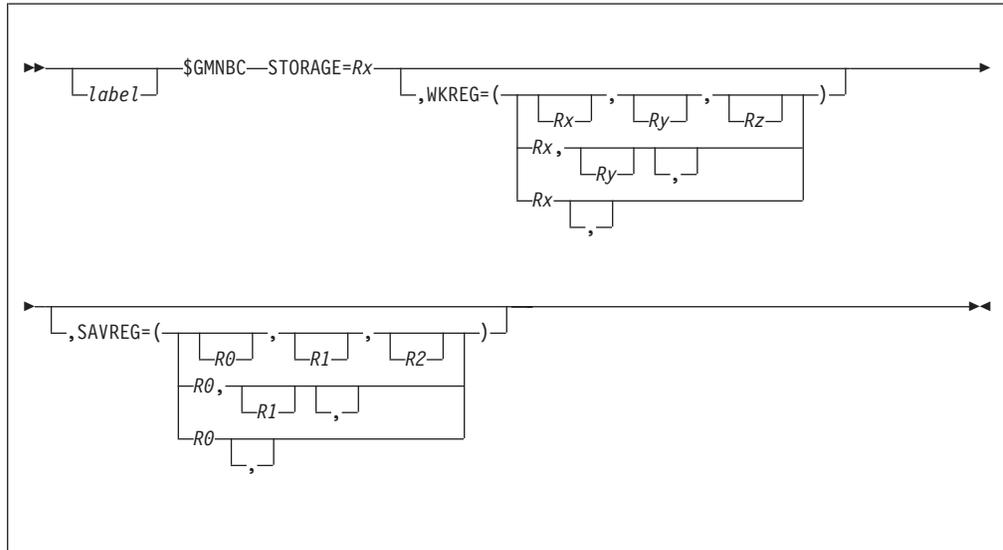
## Examples

None.

## \$GMNBC—Get Contiguous EVM Storage

Use this system macro to obtain contiguous storage in the entry control block (ECB) virtual memory (EVM).

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### STORAGE=Rx

This parameter specifies the number of contiguous 4 K byte pages to be allocated. You must use the general register R0 through R7, R14, or R15.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- This macro can be run from the entry control block (ECB) virtual memory (EVM) or the system virtual memory (SVM).
- This macro is for use in the control program (CP) only while running key 0 and in supervisor state.
- R9 must point at the ECB that will use the contiguous storage.

## Return Conditions

- On output, if there is sufficient virtual storage available, the register specified as the STORAGE parameter will contain the address of the start of the allocated storage.

**Note:** The storage acquired will be allocated in the EVM as part of the total ECB unique storage available to the ECB. It may be located above the 16 megabyte boundary. This address returned will be valid in the 31 bit addressing mode only.

- The STORAGE register will contain X'00000000' if there is insufficient virtual storage available for allocation. Control is returned to the NSI.
- When main storage blocks are exhausted, a catastrophic system error results, since insufficient real storage exists to satisfy the request. Control is not returned.
- If corruption is detected in the heap, a catastrophic system error is issued and the ECB exits immediately. Control is not returned.

## Programming Considerations

- This macro can be run on any I-stream, however it must be processed on the same I-stream specified in the CE1ISN field in the associated ECB.
- This macro is restricted to control program (CP) use only. The analogous general use macros are MALOC, CALOC, and RALOC for assembler programs and the malloc, calloc, and realloc library functions for C language programs.
- \$GMNBC storage is not contiguous in the SVM.
- The amount of virtual storage available for use as ECB unique storage is limited.
- Storage acquired by using the \$GMNBC macro is returned automatically at ECB exit (EXITC) time. Storage can also be returned at any time by using the \$RMNBC macro from the control program (CP).
- Use of the \$RMNBC macro can have a substantial negative performance impact on a TPF system in a multi-processor (tightly coupled) configuration so use of this macro should be limited.

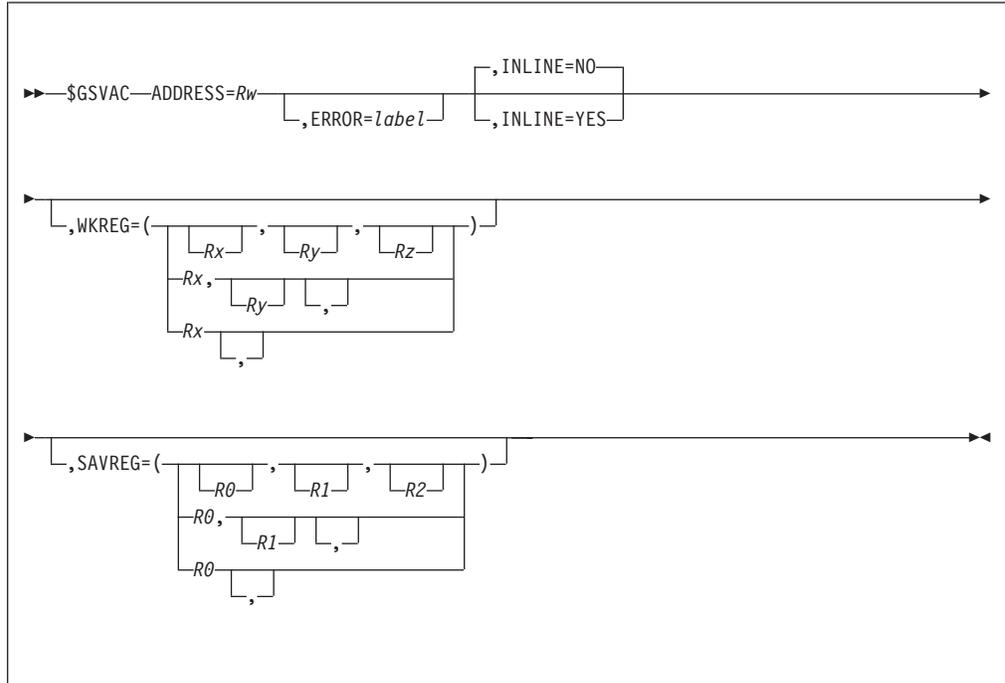
## Examples

None.

## \$GSVAC—Convert EVM Address to SVM Address

Use this system macro to convert a valid address in the entry control block (ECB) virtual memory (EVM) into the corresponding address in the system virtual memory (SVM).

### Format



#### ADDRESS=*Rw*

This parameter specifies the general register that contains the 31-bit EVM address, in the EVM of the ECB pointed to by R9, to be converted. On return, it contains the corresponding SVM address.

#### ERROR=*label*

This parameter specifies a label identifying the error routine receiving control if an error condition is raised. An error will occur when the EVM address specified in the ADDRESS parameter is invalid.

#### INLINE

This parameter specifies whether inline code or a call to the service routine is generated.

#### NO

Linkage to the service routine is generated.

#### YES

The conversion code is generated inline.

The default is NO. NO should be coded for all call invocations.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you

expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

**WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

**Entry Requirements**

- This macro can be run in either EVM or SVM.
- R9 must point to a valid ECB in the address space which is currently active.

**Return Conditions**

- Following a successful address conversion, control is returned to the next sequential instruction (NSI).
- If the address can not be converted then the ADDRESS parameter will contain the input address with the high order bit set on to indicate a conversion error. On errors, control is returned to the NSI, except when the ERROR parameter designates an error routine.
- The condition code is not preserved.
- This macro checks whether the virtual system heap address is backed by real storage. If it is not, an error is returned.

**Programming Considerations**

- This macro can be run on any I-stream, however, the ECB specified by R9 must be processed from the I-stream on which this macro is issued.
- No label is allowed on this macro.
- R13 cannot be used; it is assumed to contain the stack address.
- This macro can be run in either SVM or EVM.
- This macro will not detect all invalid addresses, and it must be noted that passing an invalid address to this routine can result in an invalid address being returned to the user without any indication from this macro. The caller should ensure only valid addresses are converted by this routine. The \$VALEC macro should be used to validate the EVA before using the \$GSVAC macro to convert the EVA to an SVA.
- If INLINE=YES is coded, the following restrictions apply:
  - R1, R2, R3 and R14 cannot be the base of the calling routine.
  - R15 cannot be used as the Frame Control Table base.
  - Must have access to CAPT tags CPMPRIVMEG, CPMPRIVEND, and CPMGMNMEG.

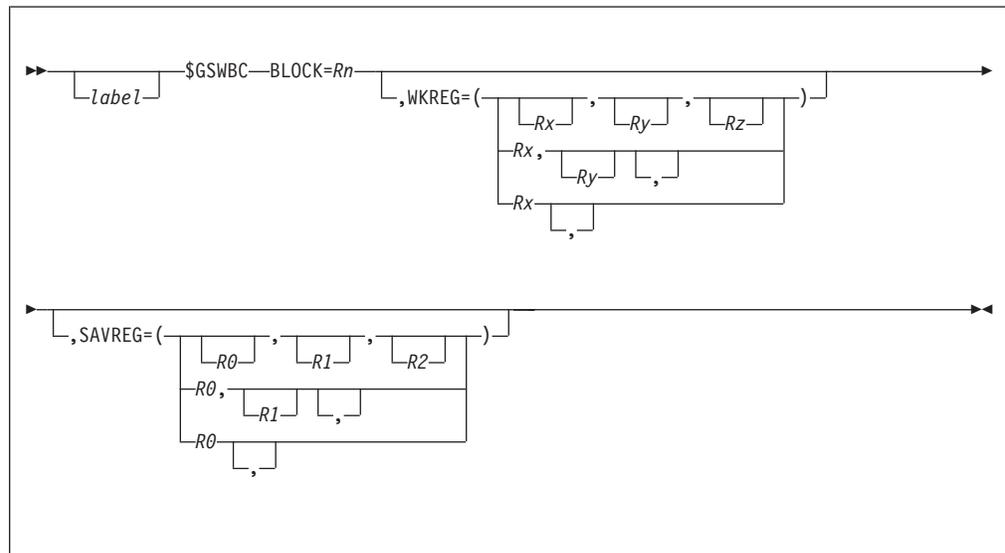
**Examples**

None.

## \$GSWBC—Get System Work Block Address

Use this system macro to obtain an available system work block (SWB) storage block address.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rn**

This output parameter contains the SWB address after macro completion.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### Entry Requirements

- The program invoking this macro must be running with a storage protection key of zero, and be in 31-bit addressing mode.
- The program can be executing in either the EVM or the SVM address space.

### Return Conditions

- Control is returned to the NSI.

- The TPF system is ended with a catastrophic error if there are no SWBs available.
- The register specified in the BLOCK parameter contains the storage block address.

## **Programming Considerations**

- This macro can be run on any I-stream.
- The address returned in the register specified by the BLOCK parameter is a 31-bit address, and may point to an SWB allocated above the 16 megabyte boundary. Code must be running in 31-bit mode to access data above the 31-bit boundary.
- System work blocks should replace I/O control blocks for all system functions. Exceptions should be made only for extremely performance-critical applications, such as DASD I/O operations.

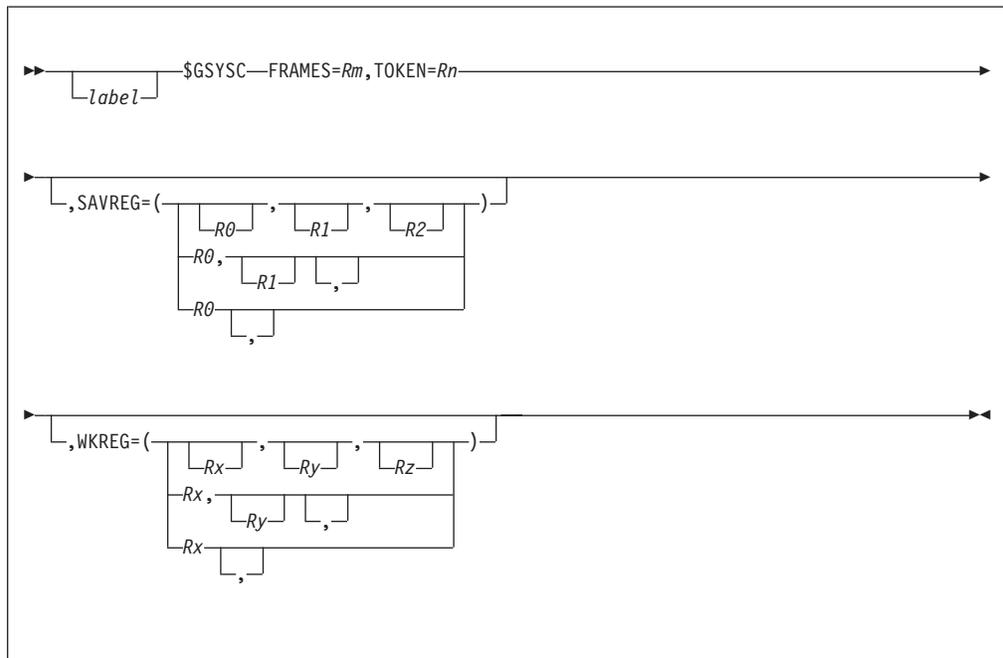
## **Examples**

None.

## \$GSYSC—Get System Heap Storage

Use this system macro to allocate a specific number of 4 KB frames as contiguous storage in the system heap.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FRAMES=Rm**

The FRAMES parameter specifies the number of contiguous 4 KB pages to be allocated. The general register used must be R0–R12, R14, or R15.

#### **TOKEN=Rn**

The TOKEN parameter specifies the address of an 8-character string that the TPF system uses to identify the allocated storage. The general register used must be R0–R12, R14, or R15.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- This macro is for use in the control program (CP) only while running in key 0 and supervisor state.
- You can use this macro in entry control block (ECB) virtual memory (EVM) or system virtual memory (SVM) address modes.

## Return Conditions

- The register specified by FRAMES contains the starting address of the allocated storage.
- The register specified by FRAMES contains 0 if:
  - The FRAMES parameter register contains zero.
  - The system heap does not contain enough storage to satisfy the allocation request.
  - The address specified by the TOKEN parameter is not in the range of addressable memory.
- Control is not returned if there is not enough real storage to satisfy the request. The TPF system issues an error and recovers through an automatic IPL.

## Programming Considerations

- This macro is restricted to CP use only.
- Real storage may not be contiguous.
- The amount of virtual storage available for the system heap is defined in CTKA.
- Storage acquired *must* be released by the RSYSC or \$RSYSC macro when it is no longer needed. If the storage is not released, it remains in use until you IPL the TPF system again.
- The address returned is a 31-bit address.
- The storage key is set to X'C'.

## Examples

The following example shows how the length of a block is converted into a number of 4 KB frames before requesting storage from the system heap. The return code is checked before trying to use the address in R14.

```

        ITUUTL REG1=R14                CONNECT WITH TABLE UPDATE DSECT
        LA    R14,ITULEN              GET THE LENGTH OF A BLOCK
        LA    R14,4095(R14)           ROUND TO THE NEXT 4 KB
        LR    R7,R14                  SAVE NUMBER OF FRAMES
        SRL   R14,12                  DETERMINE NUMBER OF 4 KB FRAMES
        LA    R6,MY_TABLE
        $GSYSC FRAMES=R14,TOKEN=R6    ALLOCATE THE STORAGE
        LTR   R14,R14                 CHECK THE RETURN CODE
        BZ    NO_STORAGE_AVAIL        BRANCH TO PROCESS ERROR
        .
        .
        routine that uses the storage
        .
        .
RELEASE_STORAGE DS 0H
        LA    R6,MY_TABLE
        $RSYSC ADDRESS=R14,FRAMES=R7,TOKEN=R6  RELEASE THE STORAGE
        LTR   R15,R15                 CHECK THE RETURN CODE
        BNZ   RELEASE_ERROR           BRANCH TO PROCESS ERROR
        .
    
```

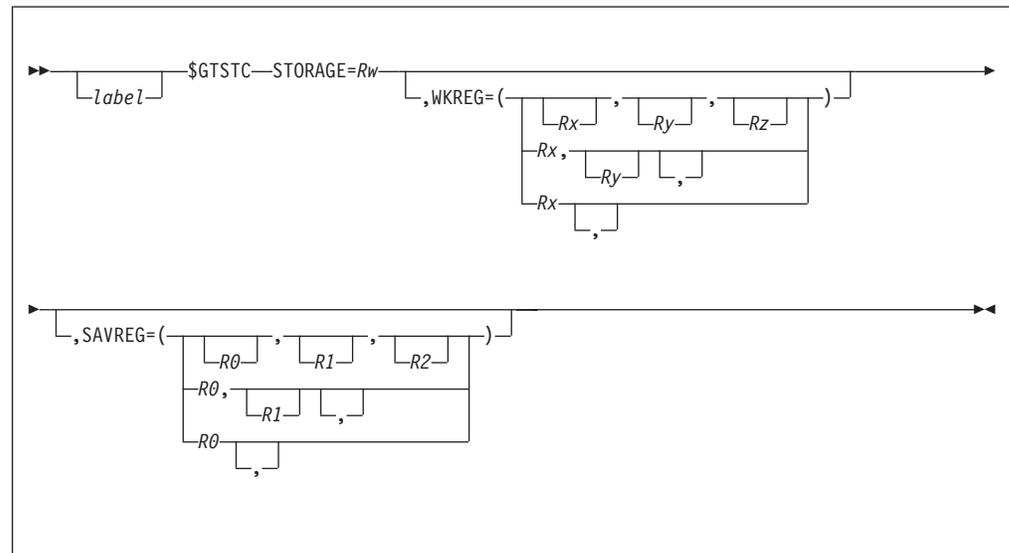
## \$GSYSC

```
      .  
      .  
MY_TABLE DC CL8'MY_TABLE'
```

## \$GTSTC—Get Contiguous EVM Stack Storage

Use this system macro to obtain contiguous storage for the ISO-C stack in the stack area of entry control block (ECB) virtual memory (EVM)

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### STORAGE=Rw

This parameter specifies the number of contiguous 4 K byte pages to be allocated. The general register used must be R0–R7, R14, R15.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### Entry Requirements

- This macro is for use in the control program (CP) only while running with protection key 0 and in supervisor state.
- This macro must be used to acquire ISO-C stack storage only. If used for acquiring storage for any other purpose, results cannot be predicted.
- R9 must point at the ECB that will use the contiguous storage.

## \$GTSTC

- This macro can be run from either EVM or from SVM.

## Return Conditions

- When this macro completes, if there is enough virtual storage available, the register specified as the STORAGE parameter contains the address of the start of the allocated storage.

**Note:** The storage acquired is allocated in the EVM as part of the ISO-C stack heap. It can be located above 16 MB. The address returned is valid in 31-bit addressing mode only.

- If the maximum amount of storage allowed for the ECB would have been exceeded by satisfying the request, a system error with exit occurs.
- When main storage blocks are used up, a catastrophic system error occurs, because insufficient real storage exists to satisfy the request. Control is not returned.

## Programming Considerations

- This macro can be run on any I-stream, however it must be processed from the same I-stream given in the CE1ISN field in the associated ECB.
- This macro is restricted to CP use only and is used only by ISO-C code to acquire initial stack allocation (ISA) storage or to extend the stack size through the stack overflow routine. Use of this macro to acquire storage for other than the ISO-C stack can corrupt the ISO-C stack.
- \$GTSTC storage is not contiguous in the SVM.
- Storage acquired using the \$GTSTC macro is automatically returned when the ECB exits.

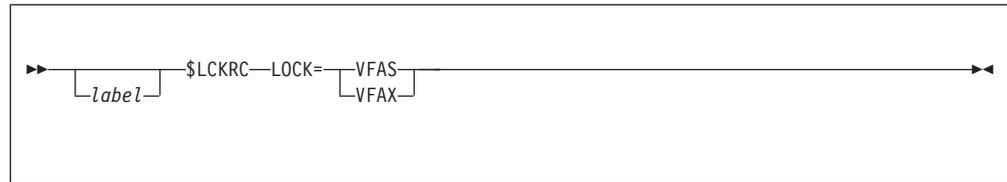
## Examples

None.

## \$LCKRC—Lock a Virtual File Access (VFA) Shared Lock or Exclusive Lock

Use this system macro to lock a virtual file access (VFA) shared lock or exclusive lock.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **LOCK**

Specifies the type of lock, where:

#### **VFAS**

Specifies shared lock.

#### **VFAX**

Specifies exclusive lock.

### Entry Requirements

Register 5 (R5) must contain the address of the VFA buffer control area (BCA) only.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Register 14 (R14) and register 15 (R15) are used to branch and link to the macro service routine so the contents of both registers are corrupted. All other registers are preserved.

### Programming Considerations

- You can run this macro from any I-stream.
- This macro is for use in the control program (CP) only.
- System errors can occur while processing the \$LCKRC request. See *Messages (System Error and Offline)* for more information about these system errors.

### Examples

The following example shows you how to lock an exclusive lock using this macro.

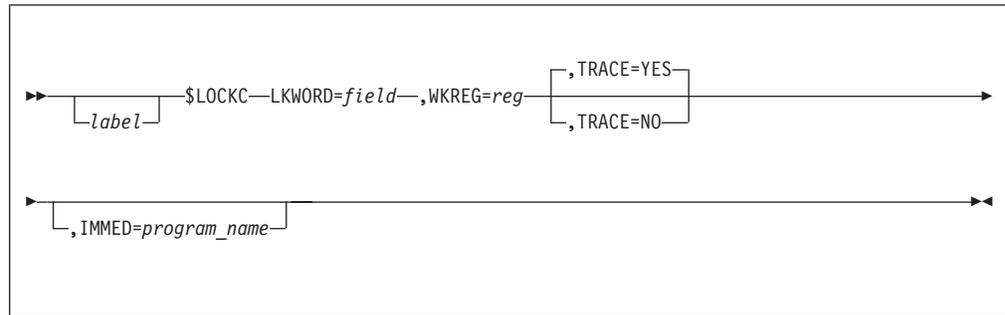
```
$LCKRC LOCK=VFAX
```

## \$LOCKC

### \$LOCKC—Lock a Resource

Use this system macro to reserve access to (or lock) a resource and prevent access to it by other I-streams. If the resource is already locked by another I-stream, the I-stream attempting the lock will *spin* (that is, loop in place) waiting for the lock to be freed. When the lock is freed, this macro locks the resource. If the spin lasts too long, the macro times out and a system error is issued.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **IMMED=program\_name**

Optional. If coded, it is the name of a routine to be given control if the lock is already held by another I-stream. This is in lieu of the spin lock.

#### **LKWORD=field**

The name of a doubleword field to be used for lock and trace functions.

#### **TRACE=YES|NO**

Specify one of the following:

##### **YES**

The current address will be placed in the second fullword of the LKWORD field.

##### **NO**

The trace function is skipped.

#### **WKREG**

The specified register is used as a work register by the macro.

### Entry Requirements

None.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of the work register specified is unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- Return is made to the next sequential instruction.
- The lock specified by LKWORD must not be held by this I-stream. If the lock is held a system error will be taken and the lock will be held.

## **\$LOCKC**

- The protection key of the program issuing the \$LOCKC macro must be the same as the item being locked.
- No other lock should be held by this I-stream since a *lock-out* condition could occur.
- If several I-streams are waiting for the same lock, it is unpredictable which I-stream will obtain the lock when it is freed, regardless of which I-stream began waiting first.
- A catastrophic system error will occur if the spin lock (IMMED is not coded) times out.

## **Examples**

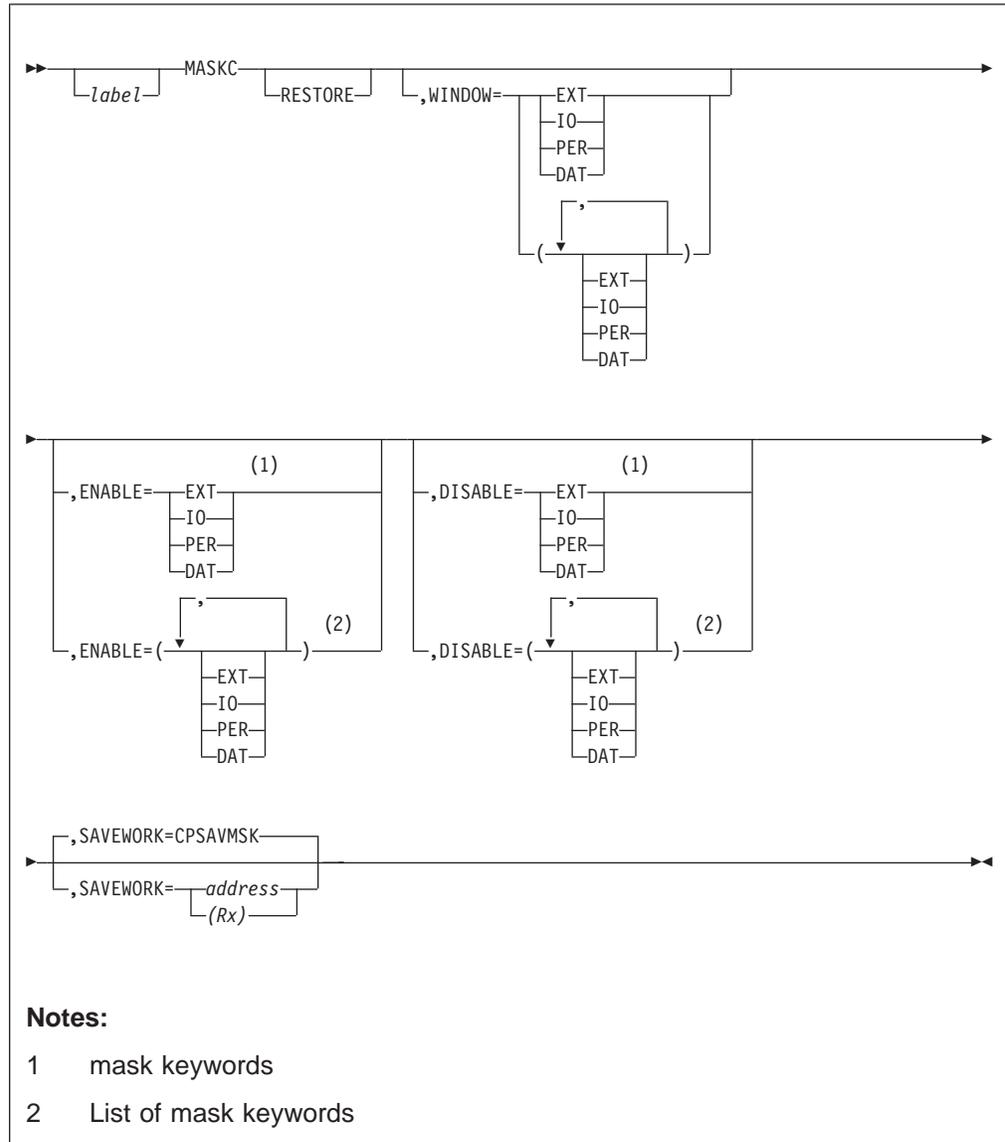
None.

## \$MASKC

### \$MASKC—Change the System Mask

Use this system macro to change fundamental characteristics of a system operation by changing the first byte of the current program status word (PSW). Input/output (I/O) interrupts, external interrupts, program event recording (PER) interrupts, and the dynamic address translation (DAT) mode may be enabled or disabled.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

**ENABLE=***mask* | (*mask*,...)

**DISABLE=***mask* | (*mask*,...)

**WINDOW=***mask* | (*mask*,...)

These parameters result in the current system mask being saved at the location specified by the `SAVEWORK` parameter and its control bits being enabled or disabled accordingly as indicated by the mask keyword (or list of mask keywords) coded.

The WINDOW parameter stores the system mask, enables the mask keywords specified, and reloads it.

The ENABLE and DISABLE parameters do not reload the system mask. Setting the bits to one/zero enables/disables the associated facilities when the system mask is reloaded by the RESTORE parameter.

These parameters specify the type of masking required with mask keywords. Masking can be specified either by individual keywords or by a list of keywords enclosed within parentheses. Either ENABLE, DISABLE, or both can be specified. The WINDOW and RESTORE parameters cannot be used on the same invocation as the DISABLE and ENABLE parameters. The RESTORE and WINDOW parameters cannot be used together on the same invocation. The mask keywords are:

**EXT**

External interrupt mask

**IO** Input/output interrupt mask

**PER**

Program event recording interrupt mask

**DAT**

Dynamic address translation mode mask. When this bit is set on, dynamic address translation is initiated. When this bit is set off, addresses are treated as real addresses.

The ENABLE and DISABLE parameters are mutually exclusive with the RESTORE and WINDOW parameters.

If a null list is specified [for example, WINDOW=()], the parameter is ignored.

**RESTORE**

Indicates a system mask should be loaded from an address specified by the SAVEWORK parameter. RESTORE is a positional parameter.

**SAVEWORK**

Specifies the 1-byte storage location for a system mask. The default location is CPSAVMSK. The address can be specified by a label or a register enclosed in parentheses. Register R0 cannot be used.

If both the ENABLE and DISABLE parameters are specified, the byte following the address specified by SAVEWORK is used as a work area.

## Entry Requirements

- The TPF system must be in supervisor state with protect key 0.
- The LOCEQ macro must have been called.

## Return Conditions

The system mask of the current PSW is set to the desired state.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is for use in the control program (CP) only.
- This macro will only change the masking of the I-stream on which it is issued.
- This macro will only change the PSW masking bits. It will not change control register contents which may also prevent certain types of interrupts.

## \$MASKC

- This macro is not visible in TPF traces.
- Specifying \$MASKC with no parameters has no effect.

## Examples

- Enabling and disabling interrupts using a list.

```
$MASKC DISABLE=(EXT,IO,PER)
```

```
<uninterruptable code>
```

```
$MASKC ENABLE=(EXT,IO,PER)
```

- Combined enabling and disabling interrupts without using a list.

The following invocations disable the PER interrupts and enable the I/O and external interrupts.

```
$MASKC DISABLE=PER,ENABLE=IO
```

```
$MASKC ENABLE=EXT
```

- Combined enabling and disabling interrupts using a list.

```
$MASKC DISABLE=(PER),ENABLE=(IO,EXT)
```

- Restoring the system mask.

– From the default location (CPSAVMSK):

```
$MASKC RESTORE
```

– From an address in register 5:

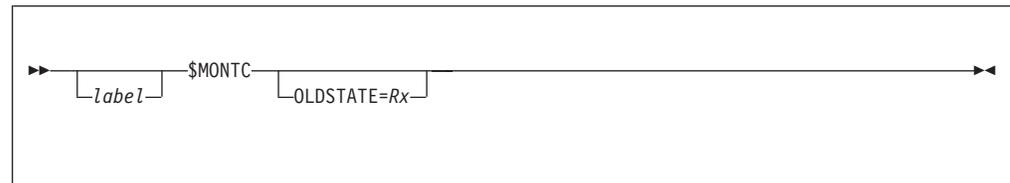
```
$MASKC RESTORE,SAVEWORK=(R5)
```

## \$MONTC–Set Supervisor State (Monitor Mode) with PSW Return

Use this system macro to change the operating state of the central processing unit (CPU) from problem to supervisor state with key 0 set and the input/output (I/O) interrupts masked. This macro provides a mechanism for returning the first part of the program status word (PSW) active at the time of its invocation to the calling program.

Supervisor state allows processing of privileged instructions. See the “MONTC–Set Supervisor State (Monitor Mode)” on page 348 macro for an E-type method of setting supervisor state.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**OLDSTATE=Rx**

An optional parameter specifying a register to receive the first word of the PSW active at the time of \$MONTC processing.

### Entry Requirements

This macro is for use in the control program (CP) only.

### Return Conditions

- Control is returned to the next sequential instruction.
- The TPF system is masked for I/O interrupts.
- The contents of all registers are preserved across this macro call except for the register specified by the OLDSTATE parameter.
- The CPU is in supervisor state (PSW bit 15 = 0).
- The storage protect key is zero (PSW bits 8 - 11).

### Programming Considerations

- This macro can be run on any I-stream.
- This macro is for system programming use only.
- Care should be exercised when operating in this state. Any location in main storage can be modified, except that to modify locations below the 512-byte line the LCPCC macro should be used first.
- Supervisor state is maintained across all other macros that can be issued by the program.

### Examples

None.

## \$MOVEC

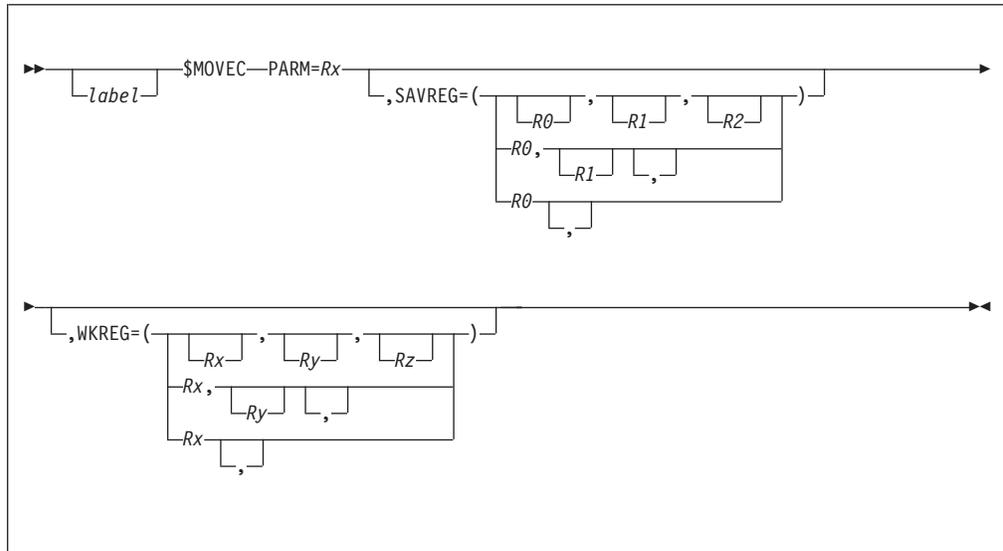
### \$MOVEC—Move Data Between EVM and SVM

Use this system macro to move:

- Data between entry control block (ECB) virtual memory (EVM) address space and system virtual memory (SVM) address space.
- Between EVM addresses that belong to two different ECBs.

This macro is similar to the E-type MOVEC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### PARM=*Rx*

This register points to the address of a parameter list in the IMOVE DSECT that contains the following fields to be filled in with the appropriate values:

#### MFROM

This 4-byte field contains the address *from which* data is moved.

#### IMTO

This 4-byte field contains the address *to which* data is moved.

#### IMSVAF

This 4-byte field contains the system virtual address (SVA) of the ECB from which the data is moved.

#### IMSVAT

This 4-byte field contains the SVA address of the ECB to which the data is moved.

#### IMLENTH

This 4-byte field contains the length of the storage to be moved.

#### IMTYPE

This 1-byte field contains the type of move to occur, either an SVA to SVA move or an SVA to ECB virtual address (EVA) move.

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or

in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

This parameter is ignored if the \$MOVEC macro is issued from ECB code.

**WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

This parameter is ignored if the \$MOVEC macro is issued from ECB code.

**Entry Requirements**

You must call the \$MOVEC macro in 31-bit addressing mode.

**Return Conditions**

Control is returned to the next sequential instruction (NSI).

**Programming Considerations**

- This macro can be run on any I-stream.
- The following combinations are valid:

From EVM to SVM	Move data from an EVM address space to an SVM address space.
From SVM to EVM	Move data from an SVM address space to an EVM address space.
From EVM1 to EVM2	Move data within an EVM address space or between different EVM address spaces.
From SVM1 to SVM2	Move data from an SVM address space to another SVM address space.

**Note:** System error dumps can occur when servicing a \$MOVEC request. See *Messages (System Error and Offline)* for more information.

**Examples**

None.

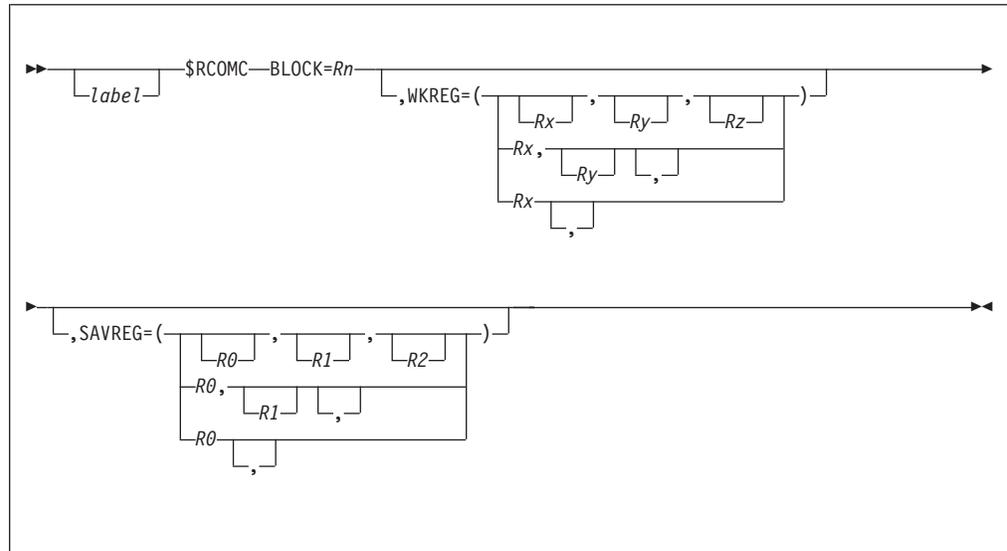
## \$RCOMC—Release Common Storage Block

Use this system macro to return:

- A common storage block to the TPF system
- Common blocks to the TPF system.

**Note:** Entry control block (ECB)-controlled programs must use the RELCC macro to return storage blocks.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rn**

This parameter is used to return the block address. The register specified must contain the 31-bit storage block address which is to be returned to the TPF system. This storage block must have been acquired through the \$GCOMC macro.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- This macro is for use in the control program (CP) only.
- The program invoking this macro must be running in a privileged mode of operation and with a storage protection key of zero. The program must also be operating in 31-bit mode. This macro operates in either the EVM or the SVM.
- R13 is assumed to contain the stack pointer.

## Return Conditions

- Control is returned to the NSI.
- The contents of the register specified in the BLOCK parameter are unpredictable.
- The condition code is not saved.

## Programming Considerations

- The \$RELBC macro can also be used to return common blocks to the TPF system. If there is any doubt whether a storage block is a common block or part of the ECB's private area, then \$RELBC should be used instead of \$RCOMC.
- This macro is for use in the control program (CP) only.
- System error dumps can occur when servicing a \$RCOMC request. See *Messages (System Error and Offline)* for more information.

## Examples

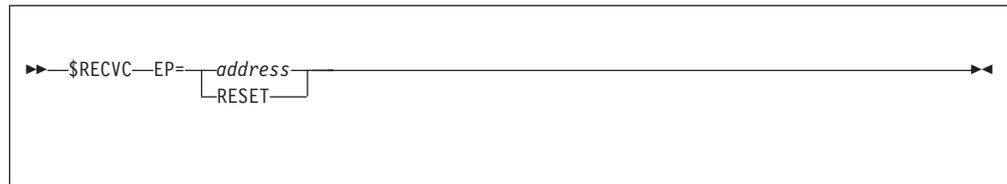
None.

## \$RECV-**Recover from Program Check**

Use this system macro to establish the location specified by the EP parameter as the routine to receive control when a program check occurs. On a program check, the general registers and system state at the time the macro was issued are restored and control is passed to the specified location.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### **EP=address|RESET**

A symbolic label specifying the entry point of the routine to receive control if a program check occurs. This label must be directly addressable.

If RESET is specified the protection set via a previous \$RECV is turned off.

### Entry Requirements

- The program must be masked for I/O and external interrupts.
- For E-type calls the program must be executing in 31-bit mode. If the program is not in 31-bit mode, authorization to use \$RECV will be refused and a program check will occur.

### Return Conditions

- For E-type calls the contents of R10 and R13 are unpredictable; all other registers are restored. For C-type calls all registers are saved and restored.

### Programming Considerations

- This macro is to be used with extreme care. The window between the macro call and the possible program check should be as small as possible. Indiscriminate use of this macro may result in catastrophic system failure.
- To eliminate the possibility of trapping unexpected errors, the calling program must be masked for I/O and External interrupts. The program must not give up control once protection is set.
- The use of this macro is restricted to system use only.
- The macro uses a monitor call (MC) to access its service routine.
- Program check protection is automatically reset when a program check occurs.
- A distinct pair of \$RECV calls must be coded for each program check expected. Only one call specifying a label is permitted prior to a program check or RESET call. Subsequent invocations of \$RECV will result in a SNAP dump, and the current process will be exited. See *Messages (System Error and Offline)* for more information.
- Protection must be reset before exiting an ECB.

- Use of this macro removes the need for the program to modify the program new PSW.

## Examples

This sample is from CSPM while servicing a SNAPC request.

```
$RECV EP=SNAPPCHK
```

```
⋮
```

```
$RECV RESET
```

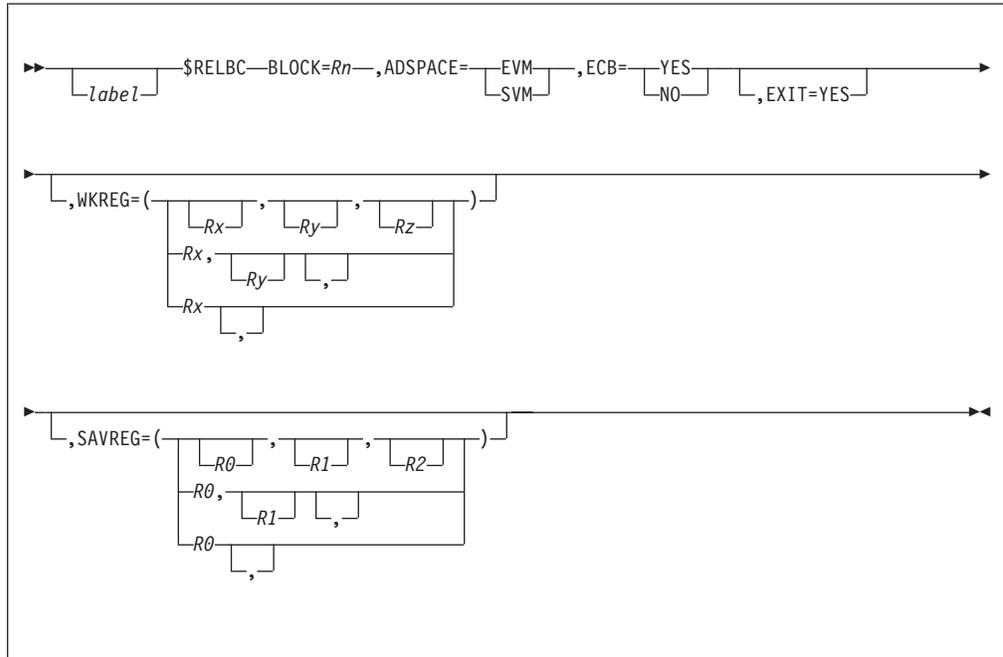
```
⋮
```

```
SNAPPCHK DS 0H
```

## \$RELBC—Release Storage Block

Use this system macro to return a storage block to the TPF system. The storage block can be any logical block or a system work block (SWB).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rn**

This parameter is used to return the block address to the TPF system. The register specified must contain the 31-bit storage block address to be returned to the TPF system. The block address returned will be correct for the address space currently being used: an EVM address if ADSPACE=EVM, an SVM address if ADSPACE=SVM.

#### **ADSPACE**

This parameter gives the address space where the calling code is operating.

#### **EVM**

The code issuing this macro is operating in the EVM on behalf of an ECB.

#### **SVM**

The code issuing this macro is operating in the SVM and may or may not be operating on behalf of an ECB.

#### **ECB**

This parameter specifies whether the block is currently connected to the ECB virtual memory (EVM).

#### **YES**

When ECB=YES, the frame containing the block must currently be connected to an ECB's virtual memory. The ECB must be on the I-stream where the \$RELBC was issued.

**NO**

The block address being returned must be an SVM address. The frame containing the block cannot be part any ECB's virtual memory.

**EXIT=YES**

EXIT=YES is a special interface for EXITC processing. During EXITC processing a check must be made for VEQR mode. If VEQR is On, all blocks must be released. If VEQR is Off, only common blocks must be released. This parameter is only valid for the EXITC service routine.

**SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

**WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- The program invoking this macro must be running in a privileged mode of operation and with a storage protection key of zero. The program must also be operating in 31-bit mode. The address space of the code issuing this macro must be correctly specified via the ADSPACE parameter.
- R13 is assumed to contain the stack pointer.

## Return Conditions

- Control is returned to the NSI.
- The contents of the register specified in the BLOCK parameter are unpredictable.
- The condition code is not saved.

## Programming Considerations

System error dumps can occur when servicing a \$RELBC request. See *Messages (System Error and Offline)* for more information about system errors.

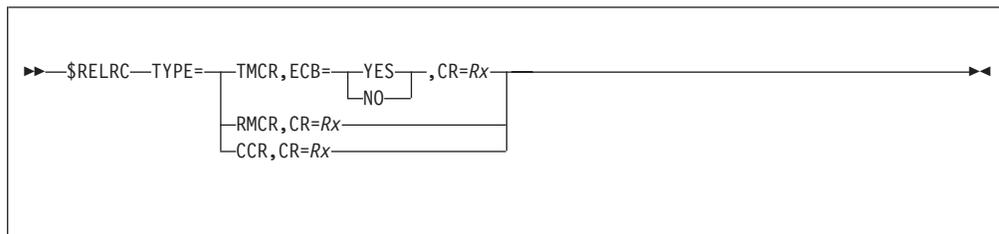
## Examples

None.

## \$RELRC – Release a Control Record

Use this system macro to release control records associated with TPF transaction services processing. This macro is only for use by the transaction manager (TM) and resource managers (RMs). Through the use of the \$RELRC macro, the TM and RMs are able to control and release the commit scope environment.

### Format



#### TYPE

Defines the type of control record to be released. This parameter is required. The following control record types are valid:

##### TMCR

Releases the transaction manager control record (TMCR).

##### RMCR

Releases the resource manager control record (RMCR) and all associated commit scope control records.

##### CCR

Releases a single commit scope control record.

#### ECB

Defines the area to which the TMCR is anchored. This parameter is required when you code TYPE=TMCR.

##### YES

Specifies that the TMCR is anchored out of the ECB at the CE2TMCR field.

##### NO

Specifies that the ECB TMCR anchor is not to be used. Instead, the TMCR address is supplied in the register specified in the CR parameter.

#### CR=R<sub>x</sub>

A register (R1–R7) that contains the address of the record to be released. This parameter is required when you code ECB=NO, TYPE=RMCR, or TYPE=CCR.

### Entry Requirements

- When you specify ECB=YES, R9 must contain the address of the ECB being processed. Control records are anchored off the ECB through the CE2TMCR field. The \$RELRC macro uses and updates this field.
- When you specify ECB=NO, the register, which is specified by the CR parameter, must contain the address of the current TMCR.
- When you specify TYPE=RMCR, the register, which is specified by the CR parameter, must contain the address of the RMCR to be released.
- When you specify TYPE=CCR, the register, which is specified by the CR parameter, must contain the address of the CCR to be released.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of the registers are preserved across this macro call with the exception of the registers specified by the macro parameters. The contents of these registers are as follows:
  - When TYPE=TMCR, the register specified on the CR parameter will contain the address of the next TMCR or zero if there are no more TMCRs.
  - When TYPE=RMCR, the register specified on the CR parameter is unchanged.
  - When TYPE=CCR, the register specified on the CR parameter is unchanged.
- When you code ECB=YES, field CE2TMCR is updated to point to the previous TMCR or zero if there is no previous TMCR.
- The condition code (CC) is not preserved across the macro call.

## Programming Considerations

- This macro can be run from any I-stream.
- This macro can only be called from control program (CP) code.
- Before releasing a TMCR, all the associated RMCRs must be released.

## Examples

```
$RELRC TYPE=TMCR,ECB=YES
```

```
$RELRC TYPE=RMCR,CR=R2
```

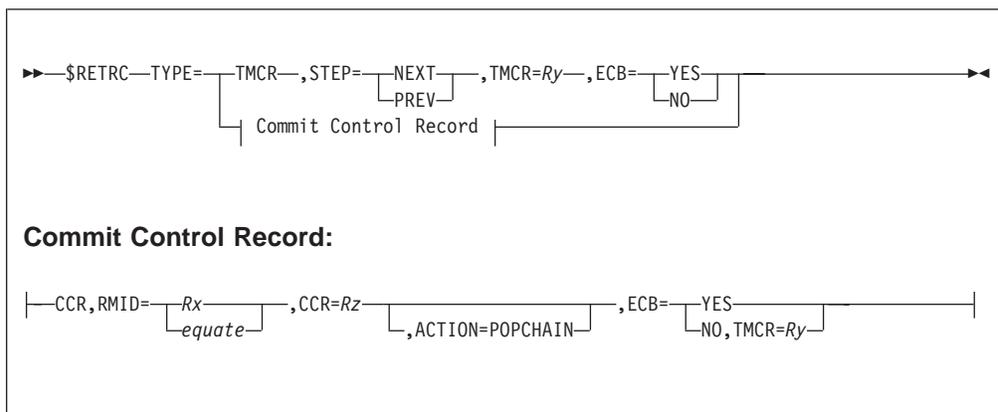
```
$RELRC TYPE=CCR,CR=R2
```

## \$RETRC

### \$RETRC – Retrieve or Modify a Control Record

Use this system macro to retrieve or modify control records that are associated with TPF transaction services processing. This macro is only used by the transaction manager (TM) and resource managers (RMs). Through the use of the \$RETRC macro, the TM and RMs control the commit scope environment.

#### Format



#### TYPE

Defines the type of control record to be acted on. This parameter is required. The following control record types are valid:

##### TMCR

Requests a transaction manager control record (TMCR).

##### CCR

Requests a commit control record (CCR).

#### TMCR=Ry

A register (R1–R7) used to identify the current TMCR and to contain the address of the retrieved TMCR. This parameter is required when you code `TYPE=TMCR` or `ECB=NO`.

#### STEP

Specifies the direction to use when retrieving a TMCR. This parameter is required when you code `TYPE=TMCR`.

##### NEXT

Retrieve the next TMCR in the chain of TMCRs.

##### PREV

Retrieve the previous TMCR in the chain of TMCRs.

#### ECB

Defines the area to which the TMCR is anchored. This parameter is required.

##### YES

Specifies that the TMCR is anchored out of the ECB at the CE2TMCR field.

##### NO

Specifies that the ECB TMCR anchor is not to be used. Instead, the TMCR address is supplied in the register specified in the `TMCR` parameter.

#### CCR=Rz

A register (R1–R7) used to identify the current CCR and to contain the address of the retrieved CCR. This parameter is required when you code `TYPE=CCR`.

**RMID=*Rx*|*equate***

A register (R0–R7) or *equate* used to define the RMID associated with the record request. This parameter is required when you code TYPE=CCR.

**ACTION**

Specifies a modification to the chaining of CCRs. This parameter is optional. When this parameter is not specified, the CCRs are retrieved.

**POPCHAIN**

The chain of CCRs is removed from the current resource manager control record (RMCR) and added to the CCR chain of the previous RMCR; that is, the chain is popped back to the previous level of nested RMCRs. POPCHAIN is only valid when you code TYPE=CCR.

**Entry Requirements**

- When you specify ECB=YES, R9 must contain the address of the ECB being processed. The current TMCR address is retrieved from ECB field CE2TMCR.
- When you specify ECB=NO, the register, which is specified by the TMCR parameter, must contain the address of the current TMCR.
- The register specified on the CCR parameter must contain the following:
  - When CCR retrieval is requested (the ACTION parameter is not coded):
    - 0 when requesting the first CCR in the chain
    - The address of the current CCR when requesting the next CCR in the chain.
  - When ACTION=POPCHAIN is specified:
    - Is not used for input but will contain the status of the operation upon return.

**Return Conditions**

- Control is returned to the next sequential instruction.
- The contents of the registers are preserved across this macro call with the exception of the registers specified by the macro parameters. The contents of these registers are as follows:
  - The register specified on the TMCR parameter will contain the following.
    - When TYPE=TMCR, the register contains:
      - The address of the retrieved TMCR.
      - 0 when the end of the chain has been reached and nothing more can be retrieved in the indicated direction.
      - –2 when a protocol error has been detected or 0 was passed as the TMCR address.
    - When TYPE=CCR, the register is unchanged.
  - The register specified on the CCR parameter contains the following.
    - When the ACTION parameter is not coded:
      - The address of the retrieved CCR.
      - 0 when the end of the chain has been reached and nothing more can be retrieved.
      - –1 when the first CCR is requested and none exists.
    - When ACTION=POPCHAIN:
      - Unchanged when the requested action is successful.
      - –2 when a protocol error has been detected and there is no previous level to pop the CCR chain back to.

## **\$RETRC**

- The register specified on the RMID parameter remains unchanged.
- The condition code (CC) is not preserved across the macro call.

## **Programming Considerations**

- This macro can be run from any I-stream.
- This macro can only be called from control program (CP) code.

## **Examples**

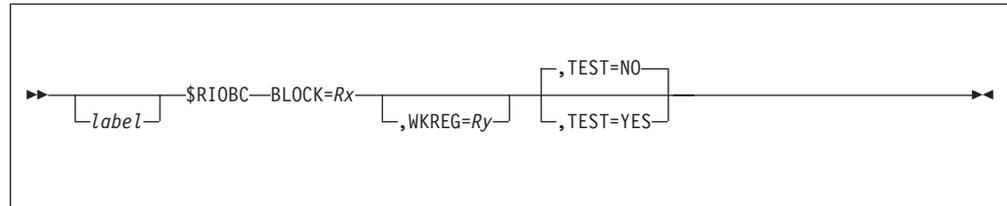
```
$RETRC TYPE=TMCR,STEP=PREV, TMCR=R1, ECB=YES
```

```
$RETRC TYPE=CCR,CCR=R1, RMID=R2, ECB=YES
```

## \$RIOBC—Release Input/Output Control Block (IOCB) Address

Use this system macro to release an input/output control block (IOCB) address.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=R<sub>x</sub>**

This register contains the address of the I/O block that is to be released.

#### **WKREG=R<sub>y</sub>**

The register specified will be used as a work register.

#### **TEST**

This parameter is used for testing and problem determination purposes.

#### **NO**

It is intended that this is used for production systems. The code to get IOBs expands in-line.

#### **YES**

This is used in a test environment to generate code to go to a central routine (see the CL\$RIOBC entry point in the CLHV segment of the CCSTOR CSECT) when releasing an IOB. It is useful in monitoring IOB list accesses. It only takes effect if the global variable SBCTEST is set on (BX'1') within this routine.

The default is NO.

### Entry Requirements

The program that invokes this macro must be running with a storage protection key of zero. It must also be operating in 31-bit mode.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The condition code is not set on return.
- The block released is no longer available for program use.

### Programming Considerations

- This macro can be run on any I-stream.
- Register R0 is used internally by the macro and should not be used to specify the address of the block or be used as a work register.
- If the TEST parameter is used in an online system, severe system degradation results due to I/O constraints.
- The TEST parameter must only be used in test environments.

**\$RIOBC**

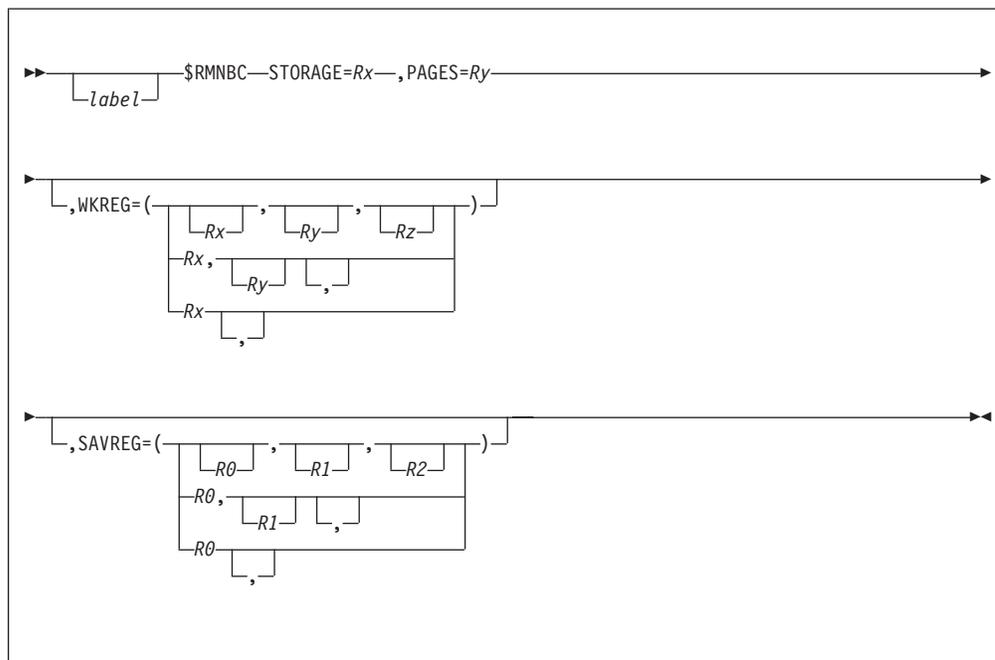
**Examples**

None.

## \$RMNBC—Release Acquired Storage

Use this system macro to return the contiguous storage acquired by the \$GMNBC macro.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **STORAGE=Rx**

This required parameter specifies the register containing the address of the storage being returned.

On input, this register contains the EVA address of an area of memory acquired by the MALOC, RALOC, CALOC, or \$GMNBC macros. This address must be on a 4 KB boundary, and must be a valid 31-bit address.

#### **PAGES=Ry**

This required parameter specifies the number of 4 KB pages to be returned to the TPF system.

On input, this register contains the number of 4 KB pages being returned. This number can be less than the number of pages originally acquired by the MALOC, RALOC, CALOC, or \$GMNBC macros.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be

## **\$RMNBC**

specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### **Entry Requirements**

- This macro can be run from the EVM or the SVM.
- This macro is for use in the control program (CP) only while running key 0 and in supervisor state.
- R9 must point at the ECB that has the contiguous storage being released.

### **Return Conditions**

- The registers specified by the STORAGE and PAGES keywords contain X'00000000'.
- Control is returned to the next sequential instruction (NSI) if the storage area being released was validly acquired via MALOC, RALOC, CALOC, or \$GMNBC.
- The ECB associated with the \$RMNBC macro is exited with a SERRC if the area being released is not valid.

### **Programming Considerations**

- This macro can be run on any I-stream, however, it must be run from the same I-stream given in the CE1ISN field in the associated ECB.
- This macro can only be called from the control program (CP) code. The analogous general use macros are FREEC for assembler programs and the (free) library function for C programs.
- Use of the \$RMNBC macro may have a substantial negative performance impact on the TPF system in a multi-processor (tightly-coupled) configuration. In general, storage acquired through the MALOC, RALOC, CALOC, or \$GMNBC macros should be returned automatically at ECB EXIT time by the TPF system, rather than explicitly by you.

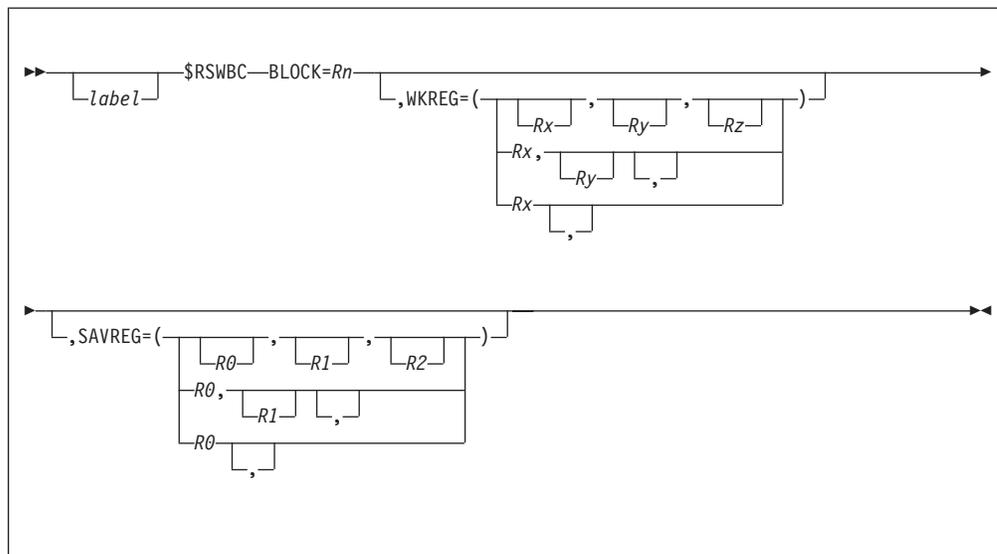
### **Examples**

None.

## \$RSWBC—Release System Work Block

Use this system macro to release an available system work block (SWB).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rn**

This register specifies the address of the SWB block to be released. This parameter is required.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### Entry Requirements

- Programs invoking this macro must be running with a storage protection key of zero and be in 31-bit addressing mode.
- Programs can be executing in either the EVM or the SVM address space.

### Return Conditions

Control is returned to the next sequential instruction (NSI).

## **\$RSWBC**

### **Programming Considerations**

- This macro can be run on any I-stream.
- \$RELBC can also be used to release SWBs.
- System error dumps can occur when servicing a \$RSWBC request. See *Messages (System Error and Offline)* for more information.

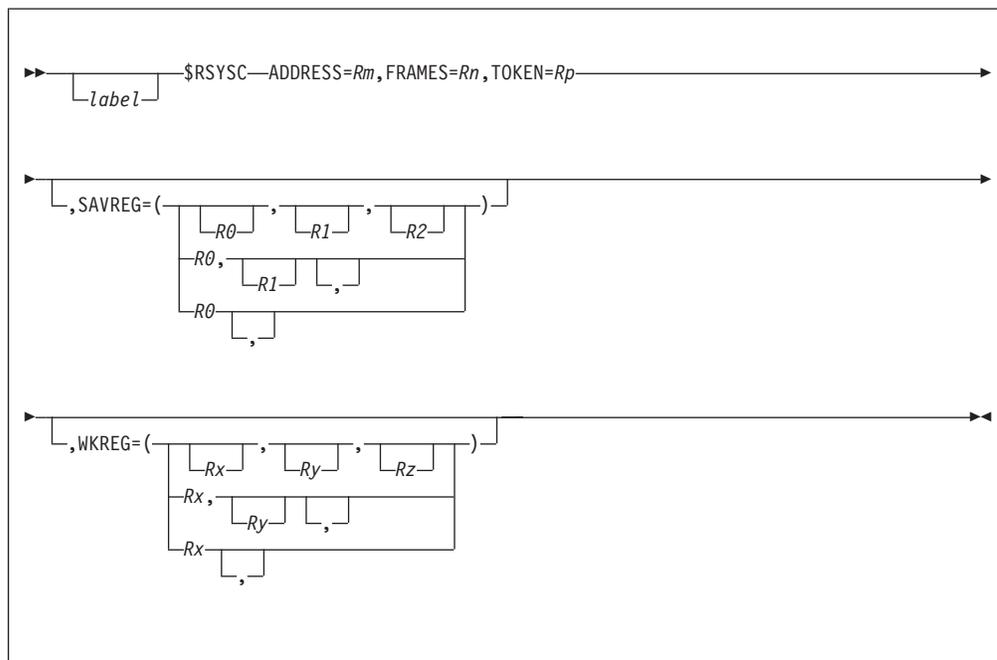
### **Examples**

None.

## \$RSYSC—Release System Heap Storage

Use this system macro to return frames to the TPF system that were allocated to the system heap.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ADDRESS=Rm**

The ADDRESS parameter specifies the register containing the starting address of the storage being returned. This address must be:

- A valid system heap address
- On a 4 KB boundary.

The general register used must be R0–R12, R14, or R15.

#### **FRAMES=Rn**

The FRAMES parameter specifies the number of 4 KB frames to be returned to the TPF system. The number of frames requested for release must be the same number of frames requested on the \$GSYSC or GSYSC macro. The general register used must be R0–R12, R14, or R15.

#### **TOKEN=Rp**

The TOKEN parameter specifies the address of an 8-character string that the TPF system uses to identify the allocated storage. This string must match the string specified on the \$GSYSC or GSYSC macro. The general register used must be R0 through R12, R14, or R15.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you

## \$RSYSC

expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- This macro is for use in the control program (CP) only while running in key 0 and in supervisor state.
- The general register indicated by ADDRESS must contain the starting address of storage allocated using the GSYSC or \$GSYSC macro.
- You can use this macro in entry control block (ECB) virtual memory (EVM) or system virtual memory (SVM) address modes.

## Return Conditions

- The TPF system returns control to the next sequential instruction (NSI).
- R15 contains the following return code values:
  - 0 The TPF system has successfully released all specified storage.

### RSYSC\_ERROR

The address specified is not valid, the storage is not in use, and the starting address, size, and token do not match current storage allocations. No storage is released.

## Programming Considerations

- This macro is restricted to CP use only.
- \$RSYSC must use the same FRAMES and TOKEN parameters specified on the GSYSC or \$GSYSC call. For example, if 12 KB of storage is allocated with a token of TABLE, the release macro must be coded to release 12 KB of storage with a token of TABLE.

## Examples

The following example shows how the length of a block is converted into a number of 4 KB frames before requesting storage from the system heap. The return code is checked before trying to use the address in R14.

```
ITUUTL REG1=R14                                CONNECT WITH TABLE UPDATE DSECT
LA     R14,ITULEN                               GET THE LENGTH OF A BLOCK
LA     R14,4095(R14)                           ROUND TO THE NEXT 4 KB
LR     R7,R14                                  SAVE NUMBER OF FRAMES
SRL   R14,12                                   DETERMINE NUMBER OF 4 KB FRAMES
LA     R6,MY_TABLE
$GSYSC FRAMES=R14,TOKEN=R6                     ALLOCATE THE STORAGE
LTR   R14,R14                                  CHECK THE RETURN CODE
BZ    NO_STORAGE_AVAIL                         BRANCH TO PROCESS ERROR
      .
      .
      routine that uses the storage
      .
```

```
RELEASE_STORAGE DS 0H  
LA R6,MY_TABLE  
$RSYSC ADDRESS=R14,FRAMES=R7,TOKEN=R6  RELEASE THE STORAGE  
LTR R15,R15  CHECK THE RETURN CODE  
BNZ RELEASE_ERROR  BRANCH TO PROCESS ERROR  
.  
.  
.  
MY_TABLE DC CL8'MY_TABLE'
```

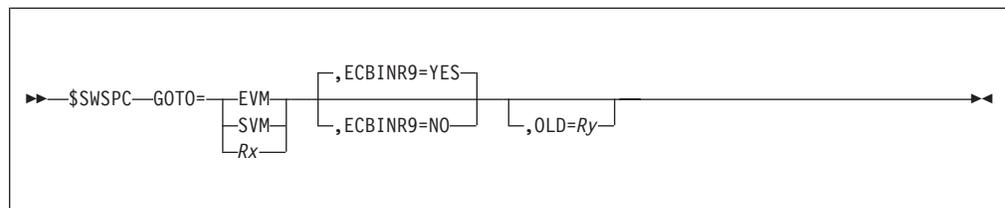
## \$SWSPC—Switch Address Space

Use this system macro to change the address space control information in the program status word (PSW). The TPF system supports 2 types of address spaces:

- Entry control block (ECB) virtual memory (EVM), which contains all memory that can be referenced or changed by the ECB. There is one EVM for each ECB in the TPF system.
- System virtual memory (SVM), which contains all storage that can be used by an I-stream.

Before using this macro, be sure you understand the effects of switching address spaces on register 9 (the ECB pointer) and control register 1 (the ECB segment and page table pointer). This macro can be run to set up these registers (with a dependency on R9) or to switch the address mode without updating the registers.

### Format



#### GOTO

This parameter identifies the address space in which the TPF system is to resume operation. There are three options:

##### EVM

The address space being switched to is the EVM. The input and return conditions vary, dependent on the ECBINR9 parameter.

If ECBINR9=YES then R9 must contain the virtual address of the ECB for the address space the ECB is executing in. On return, R9 contains the EVM address of the ECB, and control register 1 is loaded with a pointer to the ECB segment and page tables.

If ECBINR9=NO then the \$SWSPC macro will issue only the SAC instruction to switch the address space. The set-up of R9 and control register 1 would be left to the caller. This combination of parameters (GOTO=EVM and ECBINR9=NO) is essentially illogical, and is not recommended.

##### SVM

The address space being switched to is the SVM. The input and return conditions vary, dependent on the ECBINR9 parameter.

If ECBINR9=YES then R9 must contain the virtual address of the ECB for the address space the ECB is executing in. On return, R9 contains the SVM address of the ECB.

If ECBINR9=NO then the \$SWSPC macro will issue only the SAC instruction to switch the address space. If the caller plans to access the ECB or issue a \$SWSPC to the EVM, then R9 would need to be set up by the user.

*Rx* *Rx* is a register that has been previously set by the OLD option of the \$SWSPC macro. The contents of *Rx* cannot be modified by the user. When

GOTO=Rx is specified, R9 must contain the address of the ECB for the address space the ECB is executing in, unless ECBINR9=NO is coded.

**Note:** There can be no loss of control between the invocation of the \$SWSPC macro that saved the OLD register and the invocation of the \$SWSPC macro with the GOTO parameter, nor should R9 be changed.

**OLD=Ry**

This is an optional parameter. If specified, the TPF system saves the information necessary to return to the addressing mode in effect at the time the \$SWSPC macro is issued.

This parameter can be used in a subroutine to allow entry in either mode, with return to the correct addressing mode.

Ry Specify the register that will contain the return information.

**ECBINR9**

This is an optional parameter. The default is ECBINR9=YES. This parameter indicates if the macro call should use and update R9, which implies R9 is a valid ECB pointer.

**YES**

R9 will be referenced as the ECB pointer and updated to point to the ECB in the proper addressing mode. If the switch is to the EVM, then control register 1 will be loaded with the segment and page tables.

**NO**

R9 will neither be referenced nor updated. Control register 1 will not be updated when switching to the EVM.

**Entry Requirements**

- Supervisor state, protection key of zero, 31-bit addressing mode.
- See individual parameters for R9 requirements.

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- Registers are unchanged except for R9 and control register 1 as noted previously, and, optionally, the register specified by the OLD parameter.

**Programming Considerations**

- This macro can be run on any I-stream.
- The results of a \$SWSPC GOTO are totally unpredictable if there has been a loss of control between the \$SWSPC macro that saved the OLD register and the \$SWSPC macro with the GOTO parameter.
- Use of this macro can cause a performance impact.
- The usage of the ECBINR9=NO parameter is not recommended because it leaves the responsibility of maintaining R9 and control register 1 to the caller.
- If \$SWSPC GOTO=EVM,ECBINR9=YES is issued, the ECB addressed by R9 must be operating on the I-stream that issued the \$SWSPC macro.
- This macro can be run in either SVM or EVM.
- The program using this macro must be in supervisor state.
- System error dump can occur when servicing a \$SWSPC request. See *Messages (System Error and Offline)* for more information about system errors.

## \$SWSPC

### Examples

- This example will switch to the EVM, and update R9 to contain the EVA of the ECB, and control register 1 will point to the segment and page tables for that ECB address space.

**Note:** R9 must contain the valid ECB address for whatever address space the macro was invoked in upon entry.

```
$SWSPC GOTO=EVM
```

- This example will switch to the SVM, without updating R9.

**Note:** There is no dependency on R9 for this invocation.

```
$SWSPC GOTO=SVM,ECBINR9=NO
```

- This example will switch to the SVM, then switch back to the address space the first \$SWSPC switched from. R9 will contain the SVA of the ECB after the first call, and either the EVA or SVA after the second call, dependent on the address space before the first call.

**Note:** R9 must contain the valid ECB address for whatever address space the macro was invoked in upon entry.

```
$SWSPC GOTO=SVM,OLD=R3
```

```
.  
. .  
. .
```

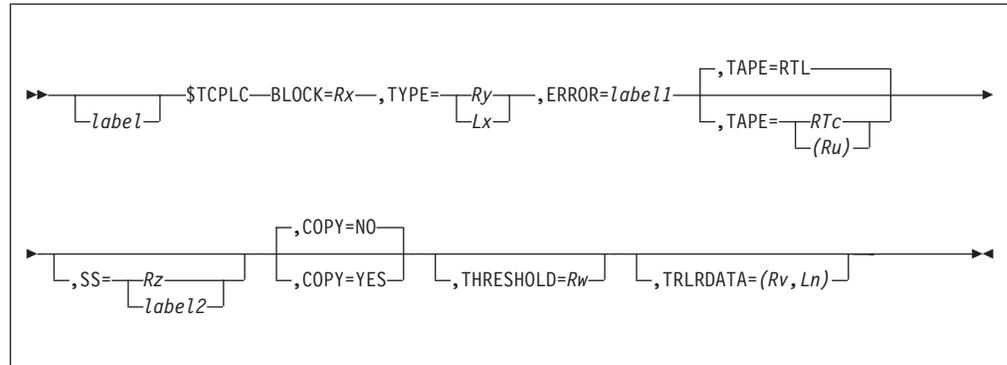
R9 and R3 must remain unaltered  
during this process ...

```
$SWSPC GOTO=R3
```

## \$TCPLC–Control Program (CP) Tape Logging

Use this system macro to log data blocks to an active real-time tape.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rx**

This parameter specifies the general register containing the system virtual address (SVA) of the block to be logged.

#### **TYPE=Ry/Lx**

This parameter specifies the type of block to be logged. A general register containing the block type or a CLH equate specifying the block type can be coded. The following types are supported:

- L0** 128 byte block
- L1** 381 byte block
- L2** 1055 byte block
- L4** 4095 byte block.

#### **ERROR=label1**

Control will be passed if the specified tape is found to be offline, not active, or the threshold value, if specified, has been exceeded.

#### **TAPE=RTL|RTc(Ru)**

This parameter specifies the symbolic tape name of a system tape to which the block is to be logged. A general register containing a pointer to the tape name may also be coded. In this case the tape name must be left-justified in the first 3 bytes of the address supplied.

The default is RTL. The default will be forced to RTA if RTL output is merged to the RTA.

#### **SS**

This parameter specifies the subsystem and subsystem user ID to which the data belongs. These IDs are in the two byte *complement and index* format. See “CEBIC–Change MDBF Subsystem/Subsystem User ID” on page 122 for more information about this format.

**Rz** When *Rz* is specified, bytes 0-1 should contain the subsystem ID, and bytes 2-3 should contain the subsystem user ID.

## \$TCPLC

*label2*

When *label2* is specified, it should point to four bytes of storage where the first two bytes contain the subsystem ID, and the second two bytes contain the subsystem user ID.

The default is the SS/SSU ID of BSS.

### **COPY**

This parameter specifies whether the data is copied into a block, and passed to the I/O routines.

### **NO**

If COPY=NO is specified, the block passed in Rx is passed to the I/O routines and returned to the TPF system at I/O completion. Therefore, the block should not be mapped in any EVM. COPY=NO is the default.

### **YES**

If COPY=YES is specified, the macro service routine gets a storage block of the type specified by the TYPE parameter, copies the data into the block, and passes the new block to the I/O routines.

### **THRESHOLD=*Rw***

This parameter specifies a register that contains the threshold value to be checked before adding the SWB to the cross list. If the number of SWBs on the module queue exceeds the threshold value, then the SWB will not be added to the cross list and the macro will return to *label1*.

### **TRLRDATA=(*Rv,Ln*)**

This parameter specifies a register, *Rv*, that contains a pointer to the user trailer data information. *Ln* specifies the length, in bytes, of the user trailer data information. This data will be written at the end of the record to be logged just before the standard trailer information. A maximum of 32 data bytes may be specified. If *Ln* is less than 32, the data is left-justified and filled with hexadecimal zeros. The standard trailer information consists of the subsystem name, subsystem user name, and the time-of-day (TOD) clock. The default is that no additional data is added to the standard trailer.

## Entry Requirements

- This macro can only be run in the system virtual memory (SVM).
- R13 must point to the system stack area.
- The caller must be in the SVM. Rx is not mapped in any EVM if COPY=NO is specified.

## Return Conditions

- Control is returned to the NSI if the specified tape is mounted, online, and the threshold value, if specified, has not been exceeded. If the tape is not mounted or is not in a useable state, control will be passed to the routine specified on the ERROR parameter.
- The condition code is not saved.
- If COPY=NO is specified, the block is no longer available for use by the calling program. If COPY=YES is specified the caller is responsible for insuring that the original block is eventually returned to the TPF system.
- A detailed indication of status of the write request is contained in R15.

X'00' An SWB has been added to the cross list. This is not an error condition.

X'04' The specified tape was not mounted or tape restart not complete.

X'08' The module queue length for the specified tape exceeded the value in the register specified by the THRESHOLD parameter.

## Programming Considerations

- This macro is for use in the control program (CP) only.
- \$TCPLC can only be invoked in the SVM.
- The \$TCPLC macro service routine ensures that tape restart is not in progress before attempting to satisfy the \$TCPLC request. If tape restart is in progress, error code 4 is returned in R15 to indicate unsuccessful completion of the request.

## Examples

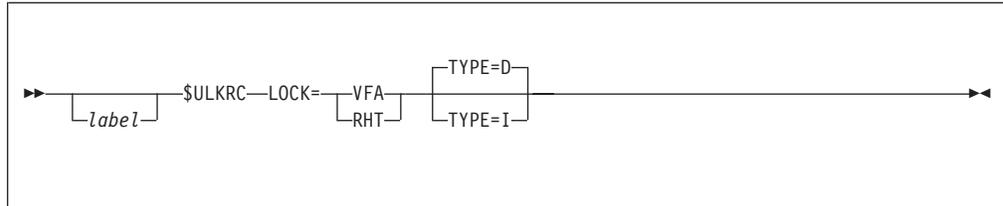
None.

## \$ULKRC

# \$ULKRC—Unlock a Virtual File Access (VFA) or Record Hold Table (RHT) Lock

Use this system macro to unlock a virtual file access (VFA) or a record hold table (RHT) lock.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### **LOCK**

Specifies the type of lock, where:

#### **VFA**

Specifies a VFA lock.

#### **RHT**

Specifies an RHT lock.

### **TYPE**

Specifies whether the lock is immediate or delayed, where:

**I** Specifies immediate.

**D** Specifies delayed. A delayed unlock is cached whenever possible, which means the lock is saved in the module file status table (MFST) and processed as part of the next channel program that communicates with the device.

## Entry Requirements

Register 5 (R5) must contain the address of the VFA buffer control area (BCA) only.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Register 14 (R14) and register 15 (R15) are used to branch and link to the macro service routine so the contents of both registers are corrupted. All other registers are preserved.

## Programming Considerations

- You can run this macro from any I-stream.
- This macro is for use in the control program (CP) only.
- System errors can occur while processing the \$ULKRC macro. See *Messages (System Error and Offline)* for more information about these system errors.

## Examples

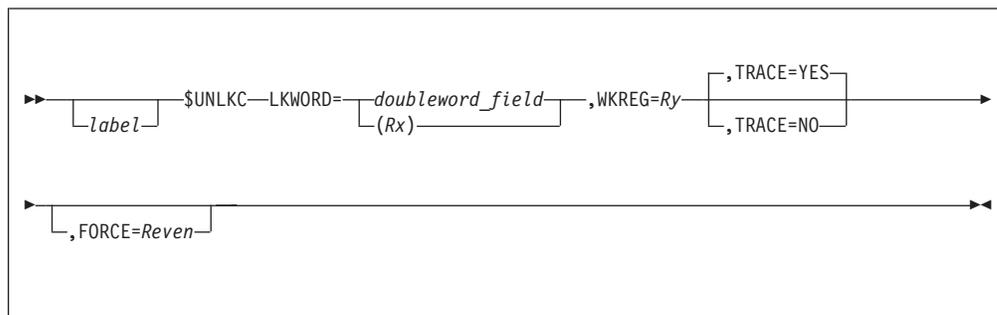
The following example shows you how to unlock a VFA lock using this macro.

```
$ULKRC LOCK=VFA
```

## \$UNLKC—Unlock a Resource

Use this system macro to unlock a resource previously locked by using the \$LOCKC macro. If the lock is not held by this I-stream, a system dump is issued and the lock is unlocked.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **LKWORLD=doubleword\_field**(*Rx*)

Doubleword field used for the lock and trace, or a base register containing the address of the doubleword field.

#### **WKREG=***Ry*

A work register to be used by this macro. If FORCE is specified, it must be the even-numbered register of an even/odd pair.

#### **TRACE=**YES|NO

If YES then the current address will be stored in the second fullword of the lock doubleword. YES is the default.

#### **FORCE=***Reven*

The even-numbered register of an even/odd pair.

If specified, FORCE is an even/odd register pair; bits 16-31 of the even register contain the I-stream number for the special unlock.

### Entry Requirements

- The resource specified by LKWORLD must be locked by the \$LOCKC macro unless FORCE is specified.
- If FORCE is specified, bits 16-31 of the even register must contain the number of the I-stream which is holding the lock.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents the specified work register are unknown. If FORCE is specified, the contents of both even/odd register pairs specified by WKREG and FORCE are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- Return is made to the next sequential instruction.
- This macro can be run on any I-stream.

## **\$UNLKC**

- The protection key of the program issuing the \$UNLKC macro must be the same as the item being unlocked.
- The lock specified by LKWORD must be held by this I-stream. If the lock is not held a system error will be taken unless the FORCE parameter is coded.

## **Examples**

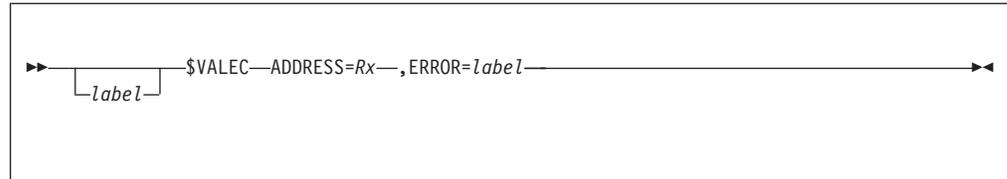
None.

## \$VALEC—Validate Entry Control Block (ECB) Virtual Memory (EVM)

### Address

Use this system macro to generate inline code that uses an entry control block (ECB) virtual memory (EVM) segment and page tables to determine if an EVM address is valid.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ADDRESS**

This required parameter specifies the register containing the EVM address to be validated.

*Rx* Specifies a general purpose register, from R0 through R7.

#### **ERROR=*label***

This required parameter specifies a program label, in the calling program, where control is to be transferred in the event the specified EVM address is not valid.

### Entry Requirements

- The invoking routine must be executing in the SVM and be in supervisor state.
- Register 9 must contain the SVM address of the target ECB.
- Register 11 must contain the standard TPF address value (X'1000').

### Return Conditions

- The register specified in the ADDRESS parameter is unchanged.
- The contents of registers R14 and R15 are unpredictable.
- When validation is successful, control is returned to the next sequential instruction (NSI) following the macro. No address conversion was performed.
- When validation is unsuccessful, processing continues at the address specified by the ERROR parameter.

### Programming Considerations

- This macro is for use in the control program (CP) only.
- The literals generated by this macro must be accessible to the generated code.
- This macro uses general registers 14 and 15. They are not saved prior to being used.

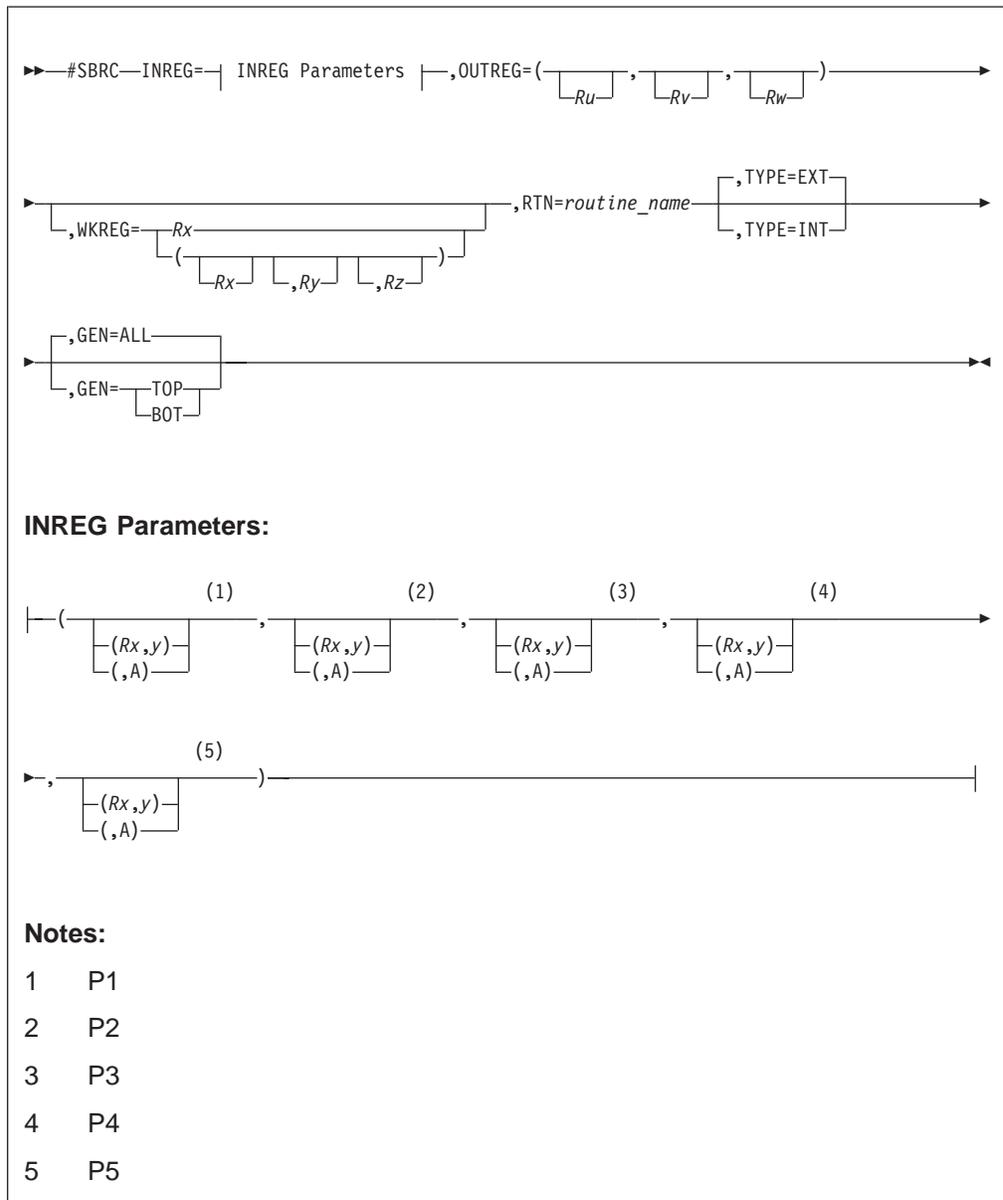
### Examples

None.

## #SBRC—Standard Linkage Macro Subroutine

Use this system macro to generate the most efficient standard linkage for the macros that call it.

### Format



### INREG=(P1,P2,P3,P4,P5)

This parameter contains positional sublist entries P1, P2,...,P5 for general registers R14, R15, R0, R1, and R2 respectively. There is direct mapping between the position of the sublist entry and the general register to which it refers (for example, P1 refers to R14, P2 refers to R15, and so on).

The format of a sublist entry is (Rx,y) or Null.

- The first parameter of a sublist entry is a general register name, R0 thru R15, or null.

- The second parameter is the character A or null. A signifies that the contents of the corresponding register in the range R0 – R2 is an applicable parameter to the subroutine and should be initialized with the contents of the register Rx. If Rx is not the actual corresponding register, then the contents of the register to which Rx refers are saved.

**Notes:**

1. Registers R14 and R15 are always saved, and their sublist entries P1 and P2 cannot contain Rx parameters. This is because only registers R0-R2 can contain parameters to routines. The INREG parameter is also used by #SBRC to reload any corrupted registers after return from the service subroutine. However, only registers R0, R1 and R2 are considered here since registers R14 and R15 will always be reloaded on return by the service subroutine.
2. The format of a sublist entry for registers R14 and R15 can only be (,A) or Null because these registers are always used as part of the basic subroutine linkage, and are always saved.

For another example,

```
INREG=( ,(,A), (R6,A) ,, (R7,))
```

generates:

```
STM R14,R0,SAVAREA
LR R0,R6
LR R2,R7
```

This means R15 and R0 must be saved over this subroutine call, and, R0 and R2 are the inputs to this subroutine through R6 and R7 respectively.

The defaults for each sublist entry are the corresponding mapped registers and null.

**OUTREG=(P1,P2,P3)**

This parameter deals with initializing registers on return from the service subroutine. It can contain a maximum of 3 registers with P0, P1, and P2 as positional entries for registers R0, R1, and R2 respectively. The delimiter for each position is a comma.

For example, when a subroutine returns R0 and R2 with resultant data, and the calling program expects the results to be contained in registers R6 and R0 respectively, the following #SBRC parameter is formatted. For example,

```
OUTREG=(R6,,R0)
```

generates:

```
LR R6,R0
LR R0,R2
```

The default for each sublist entry is null, which generates no code.

**RTN=routine\_name**

This parameter contains the name of the routine to be invoked. This parameter is mandatory and must not be null. See each individual macro for the specified routine.

**WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used

## #SBRC

here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### TYPE

This parameter determines the way in which the RTN parameter is resolved.

### EXT

The RTN parameter is assumed to specify an external routine, RTN=EXT is the default.

### INT

The RTN parameter is assumed to contain a routine that is directly addressable.

### GEN

This parameter defines the code generation. There are two options; GEN=ALL, or GEN=TOP followed by a parameter list and GEN=BOT.

### ALL

The complete linkage is generated. This is the default.

### TOP|BOT

When GEN=TOP is coded, the code is generated through the BASR. This is followed by an inline parameter list, which is followed by GEN=BOT.

When GEN=TOP and GEN=BOT are coded, the other parameters coded on the #SBRC macro must be identical.

For example:

```
#SBRC GEN=TOP,INREG=(,A),(R6,A),,(R7,),OUTREG=(R6,,R0),RTN=routine,  
TYPE=EXT DC A(PARMLIST)  
#SBRC GEN=BOT,INREG=(,A),(R6,A),,(R7,),OUTREG=(R6,,R0),RTN=routine,  
TYPE=EXT
```

## Entry Requirements

- R13 is assumed to contain the stack pointer. The macro uses a predefined area in the stack to save and restore R14 through R2 over the macro call.
- See the entry requirements listed for each of the individual calling macros.

## Return Conditions

- Control is returned to the next sequential instruction in all cases.
- The contents of R14 and R15 will unchanged upon return.
- The condition code is not altered by the #SBRC macro, and is passed through to the exit of the macro invoking #SBRC.
- See the individual calling macros for the specific return conditions associated with each macro.

## Programming Considerations

- #SBRC is an internal macro subroutine. Some of the macros that call it are:
  - The storage block control macros are GETBC, GTMBC, RELBC, and RLMBC
  - The storage block inquiry macros are BLKBC, CLHCC, CLHEC, NUMBC, NXTBC, VALBC, and VALTC
  - The dispatch control macros are ADDFC, ADDLC, STPLC, and STTLC

- The dispatch inquiry macros are NUMLC and NXTLC.

The name #SBRC deviates from the standard naming convention to indicate this differentiation.

- See the individual calling macros for specific programming considerations associated with each macro.

## Examples

None.

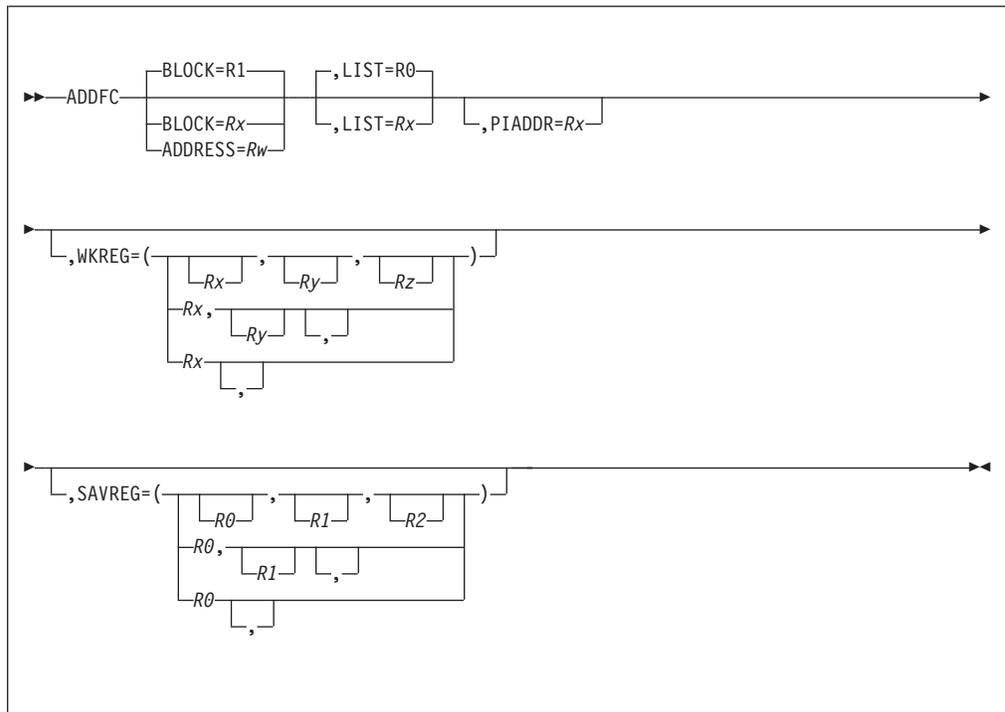
## ADDFC—Add a Block to the Top of a Dispatch List

Use this system macro to add a block to the top of the specified system task dispatcher list.

The block added through this macro will be the first block dispatched the next time the specified system task dispatcher list is processed.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### **BLOCK=R1|*R<sub>x</sub>***

The register specified on this parameter contains the EVM address of the storage block to be added to the specified system task dispatcher list. If the parameter is omitted, the default assignment is R1.

The ADDRESS and BLOCK parameters are mutually exclusive.

#### **ADDRESS=R<sub>w</sub>**

The register specified on this parameter contains the SVA address of the storage block to be added to the specified System Task Dispatcher list as indicated by the LIST parameter. There is no default.

The ADDRESS and BLOCK parameters are mutually exclusive.

#### **LIST=R0|*R<sub>x</sub>***

The register specified on this input parameter contains a dispatch list equate value as defined in the CLHEQ macro. If the parameter is omitted, the default assignment is R0.

Valid equate values are:

**#CLHRDY**  
Ready List

**#CLHINP**  
Input List

**#CLHDEF**  
Defer List

#### **PIADDR=*R*x**

The register specified on this parameter contains the address of the postinterrupt routine to be given control when dispatched from the list. If the parameter is omitted, it is assumed that displacements 6-7 in the specified block will contain the postinterrupt address.

#### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

#### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

- If the macro is coded in a real-time segment, R9 must contain the address of the ECB being processed.
- If the macro is coded in the control program (CP), the routine invoking this macro must be running in a privileged mode of operation and with a storage protect key of zero.
- The block address passed in the BLOCK parameter must be a valid EVM address and the block address passed in the ADDRESS parameter must be a valid SVM address.
- The ADDRESS and BLOCK parameters are mutually exclusive. If neither is coded, the default is BLOCK=R1.
- If the PIADDR parameter is not used, the postinterrupt status halfword at relative location 6 and 7 in the block must contain a valid branch vector address.

## Return Conditions

- Control is returned to the next sequential instruction.
- When invoked from a real-time segment, the contents of all user registers are preserved across this macro call. When invoked from the control program (CP), the registers specified on the WKREG parameter will be used to save the contents of any *volatile* registers specified on the SAVREG parameter. All other registers will be returned intact.
- The condition code is not saved across this macro call.

## ADDFC

### Programming Considerations

- This macro can be run on any I-stream.
- When called from a real-time segment and using the BLOCK parameter, ADDFC will disconnect the block from the ECB private area. Upon return the real-time segment must clear the block held indicator, from the core block reference word, to prevent double release problems in EXITC processing.

When the ADDRESS parameter is used instead of the BLOCK parameter, it is assumed that the block is not attached and no attempt is made to disconnect it.

- This macro cannot be used to add a block to the cross list. To add to the cross list, use the \$ADPC macro.
- The WKREG and SAVREG parameters are used only on macro invocations from the control program (CP). These parameters have no effect on the macro expansions in real-time programs.

### Examples

None.



## ADDLC

### #CLHINP

Input List

### #CLHDEF

Defer List

### PIADDR=*R*<sub>x</sub>

The register specified on this parameter contains the address of the postinterrupt routine to be given control when dispatched from the list. If the parameter is omitted, it is assumed that displacements 6-7 into the specified block will contain the postinterrupt address.

### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### DEBUG=NO|YES

This parameter is only valid on calls from the control program (CP). If coded as YES, the value of all macro parameters at the time of the macro call will be printed. This parameter is not for general use, and will not effect operation at processing time.

## Entry Requirements

- If the macro is coded in a real-time segment, R9 must contain the address of the ECB being processed.
- If the macro is coded in the control program (CP), the routine invoking this macro must be running in a privileged mode of operation and with a storage protect key of zero.
- The block address passed in the BLOCK parameter must be a valid EVM address. The block address passed in the ADDRESS parameter must be a valid SVM address.
- The BLOCK and ADDRESS parameters are mutually exclusive. If neither is coded, the default is BLOCK=R0.
- If the PIADDR parameter is not used, the postinterrupt status halfword at relative location 6 and 7 in the block must contain a valid branch vector address.

## Return Conditions

- Control is returned to the next sequential instruction.
- When called from a real-time segment, the contents of all user registers are preserved across the macro call. When invoked from the control program (CP),

the registers specified on the WKREG parameter will be used to save the contents of any *volatile* registers specified on the SAVREG parameter. All other registers will be returned intact.

- The condition code is not saved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- When ADDLC is called from a real-time segment and the BLOCK parameter is used, ADDLC disconnects the block from the ECB's private area. Upon return the real-time segment must clear the block held indicator, from the core block reference word, to prevent double release problems in EXITC processing.  
When the ADDRESS parameter is used instead of the BLOCK parameter, it is assumed that the block is not attached and no attempt is made to disconnect it.
- This macro cannot be used to add a block to the cross list. To add to the cross list, use the \$ADPC macro.
- The WKREG and SAVREG parameters are only used on macro invocations from the control program (CP). These parameters have no effect on the macro expansions in real-time programs.
- See the prologue of the #SBRC macro for more information about the WKREG and SAVREG parameters.

## Examples

None.



## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- The BBEWP macro can be used only by recoup segments.
- The ECB is used and modified by this macro.

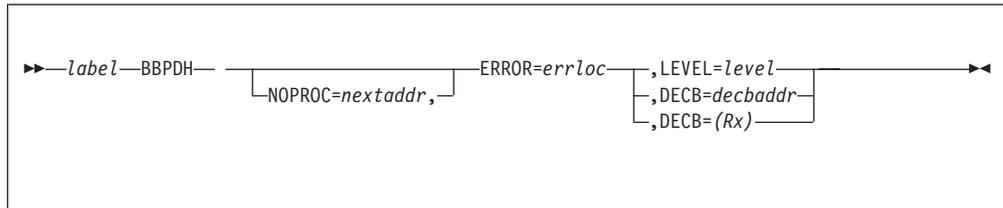
## Examples

```
BBEWP ITEM=EBW008,LEVEL=2  
BBEWP ITEM=CURR
```

## BBPDH–Recoup Record Find and Count Interface

Use this system macro to provide settings that are used by the recoup and program directory update (PDU) online pool directory handler (BPDH) and the record ID update table (BRID or BRV7).

### Format



*label*

is a symbolic name that can be assigned to the macro statement.

#### **NOPROC=nextaddr**

specifies that the address being chain chased is an RCI candidate that should not be processed if it has already been chain chased, where *nextaddr* is the location in which to branch if the address has already been chain chased. If the NOPROC parameter is not specified, the address will be chain chased even if it has already been chased.

#### **ERROR=errloc**

specifies the location in which to be branched if an error is detected by BPDH, where *errloc* is the location in which to branch.

#### **LEVEL=level**

specifies the level on which to run the FINDC macro in the TPF Database Facility (TPPDF) recoup environment, where *level* is a number from 0 to 7.

**Note:** Specify the LEVEL parameter when the entry control block (ECB) level is passed from the TPDFDF product.

#### **DECB**

specifies one of the following:

##### *decbaddr*

specifies a core location containing the address of a data event control block (DECB) that will be used in the processing of a FINDC macro.

##### *(Rx)*

specifies a general register (R1–R7) containing the address of a DECB that will be used in the processing of a FINDC macro.

### Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- The LEVEL parameter must be specified if the BBPDH macro is called from the TPDFDF product.
- Registers R14 and R15 must be available for use. Their contents are not saved.
- Label EBW061 in the ECB contains a recoup restart area index. When the DECB parameter is coded, the item containing the file address to be processed is located in the recoup stack area (BC0SA)

## Return Conditions

- If an error occurs while interfacing with BPDH, the BBPDH macro branches to the location specified with the ERROR parameter. Otherwise, control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- The BBPDH macro can only be used by recoup segments.
- The BBPDH macro uses and modifies the ECB.
- The BBPDH macro enters BPDH to interface with the recoup pseudo directories.
- The BBPDH macro enters BRID to interface with the recoup record ID counts table.
- When DECB=(Rx), general registers R1 to R7 are valid.
- The BBPDH macro enters BRV7 to handle alternate ID processing.

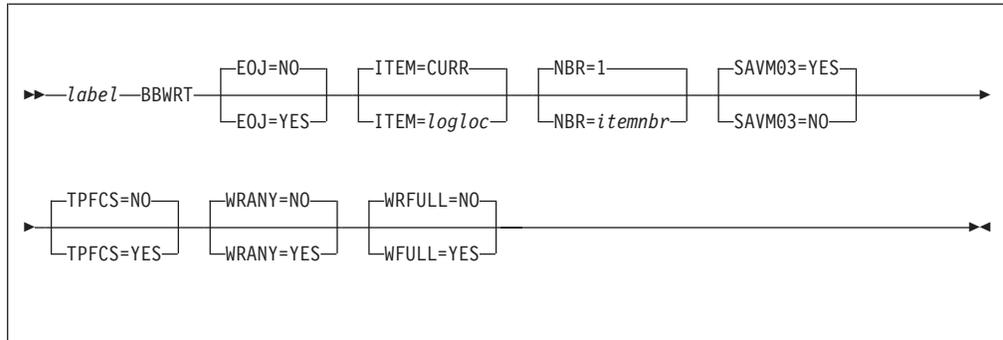
## Examples

```
BBPDH ERROR=ERRLOC,LEVEL=0,NOPROC=DUPLOC  
BBPDH ERROR=ERRLOC,DECB=(R3)  
BBPDH ERROR=ERRLOC,DECB=BCØRDCB
```

## BBWRT–Recoup Logging Item Setup

Use this system macro to place recoup items in the RCP tape logging block.

### Format



*label*

is a symbolic name that can be assigned to the macro statement.

#### EOJ

specifies the end-of-job (EOJ) indicator.

#### NO

specifies that the end of job has not been reached and the calling segment is passing items to be written to the RCP tape.

#### YES

specifies that the end of job has been reached and there are no more items to be written to the RCP tape.

#### ITEM

specifies one of the following:

#### CURR

specifies that the location of the recoup logging item is the current slot in the recoup save area (BC0SA).

*logloc*

specifies the work location of a recoup logging item.

#### NBR=*itemnbr*

specifies the number of items starting at the location specified with the ITEM parameter, where *itemnbr* is a number greater than 0.

#### SAVM03

specifies whether to save the previous EBCM03 value and restore this value across this macro call.

#### YES

saves and restores the previous EBCM03 value across the macro call.

#### NO

does not save and restore the previous EBCM03 value across the macro call.

#### TPFCS

specifies whether the recoup logging item is a TPF collection support (TPFCS) item.

**YES**

specifies the recoup logging item is a TPFCS item.

**NO**

specifies the recoup logging item is not a TPFCS item.

**WRANY**

specifies whether the current RCP tape block is written if there are any items in the block but no current item is passed.

**NO**

does not write the current RCP tape block if there are any items in the block but no current item is passed.

**YES**

writes the current RCP tape block if there are any items in the block but no current item is passed.

**WRFULL**

specifies whether the current RCP tape block is written if the block is completely full.

**NO**

does not write the current RCP tape block if the block is completely full.

**YES**

writes the current RCP tape block if the block is completely full.

## Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- The recoup logging item must be formatted according to the BCOSA data structure.
- The recoup logging item must be set up in the current slot in the recoup stack save area unless the item is specified in another work location.
- Registers R14 and R15 must be available for use. Their contents are not saved.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of registers R0–R7 are preserved across this macro call.

## Programming Considerations

- The BBWRT macro can be used only by recoup segments.

## Examples

BBWRT

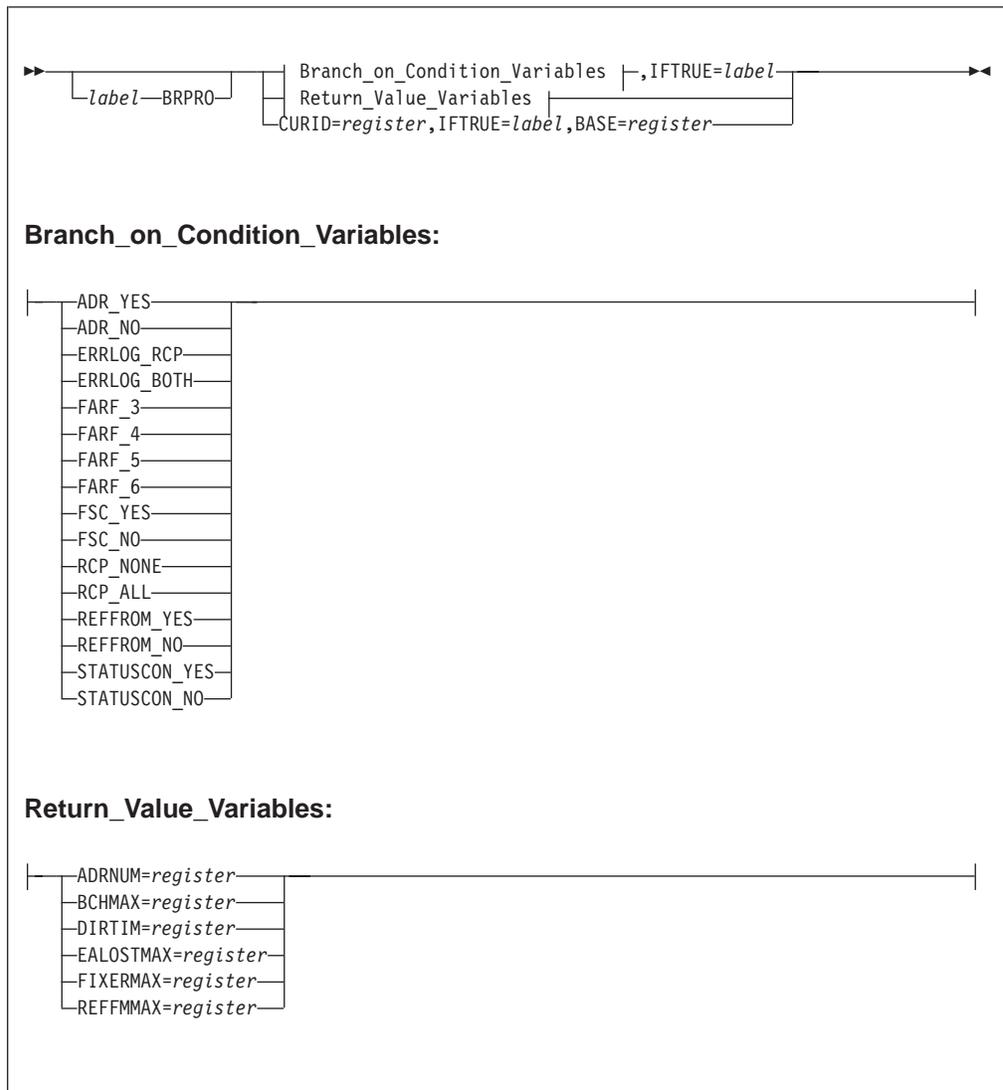
BBWRT ITEM=CURR

BBWRT ITEM=EBW000,NBR=2,SAVM03=NO,E0J=YES

## BRPRO–Query Recoup Options

Use this system macro to query the status of current recoup run options that are held as bit settings at BK0APIS in the recoup keypoint (BK0RP). These recoup run options are set by using the ZRECP PROFILE command. See *TPF Operations* for more information about recoup run-time options and the ZRECP PROFILE command.

### Format



*label*

is a symbolic name that can be assigned to the macro statement.

**IFTRUE=*label***

specifies a user-defined label in which to branch if a condition is true.

**CURID=*register***

specifies a register that contains the record ID of a record that you want to compare to the recoup keypoint record. The register specified with the BASE parameter points to the recoup keypoint.

**Notes:**

1. Register 14 (R14) and R15 must **not** be used for this parameter.
2. Data level D must contain the save area (BC0SA) block.

**BASE=register**

specifies a register that contains the address of the recoup keypoint.

**ADR\_YES**

checks the status of the ADR\_YES option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**ADR\_NO**

checks the status of the ADR\_NO option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**ERRLOG\_RCP**

checks the status of the ERRLOG\_RCP option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**ERRLOG\_BOTH**

checks the status of the ERRLOG\_BOTH option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FARF\_3**

checks the status of the REFFROM\_FARF3 option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FARF\_4**

checks the status of the REFFROM\_FARF4 option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FARF\_5**

checks the status of the REFFROM\_FARF5 option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FARF\_6**

checks the status of the REFFROM\_FARF6 option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FSC\_YES**

checks the status of the FSC\_YES option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**FSC\_NO**

checks the status of the FSC\_NO option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**RCP\_NONE**

checks the status of the RCP\_NONE option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**RCP\_ALL**

checks the status of the RCP\_ALL option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**REFFROM\_YES**

checks the status of the REFFROM\_DEACTIVATION option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

**REFFROM\_NO**

checks the status of the REFFROM\_DEACTIVATION option bit and, if the option bit is set off, branches to the label specified by the IFTRUE parameter.

## BRPRO

### **STATUSCON\_YES**

checks the status of the STATUSCON option bit and, if the option bit is set on, branches to the label specified by the IFTRUE parameter.

### **STATUSCON\_NO**

checks the status of the STATUSCON option bit and, if the option bit is set off, branches to the label specified by the IFTRUE parameter.

### **ADNUM=register**

returns the ADNUM value in the specified register.

### **BCHMAX=register**

returns the BCHMAX value in the specified register.

### **DIRTIM=register**

returns the directory capture timeout (DIRTIM) value in the specified register.

### **EALOSTMAX=register**

returns the EALOSTMAX value in the specified register.

### **FIXERMAX=register**

returns the FIXERMAX value in the specified register.

### **REFFMAX=register**

returns the REFFMAX value in the specified register.

## Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- The recoup keypoint (BKORP) has a general register pointing to it so this macro can use the labels in the keypoint.
- Scratch registers R14 and R15 must be available for use. Their contents are preserved across this macro call.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of all registers are preserved across this macro call.

## Programming Considerations

- The BRPRO macro can be used only by recoup segments.
- Use the BRPRO macro only on the main I-stream.

## Examples

```
BRPRO DIRTIM=R15
BRPRO CURID=R6,IFTRUE=CURLLOC,BASE=R3
BRPRO ADR_YES,IFTRUE=GENLOC
```



## BRSTR

- The contents of R14 and R15 are unknown. The contents of registers R0–R7 are restored from the recoup stack save area unless you specify REG=N in which case the current contents of R0–R7 remain unchanged across the macro call.

## Programming Considerations

- This macro is used in conjunction with BSAVE.

## Examples

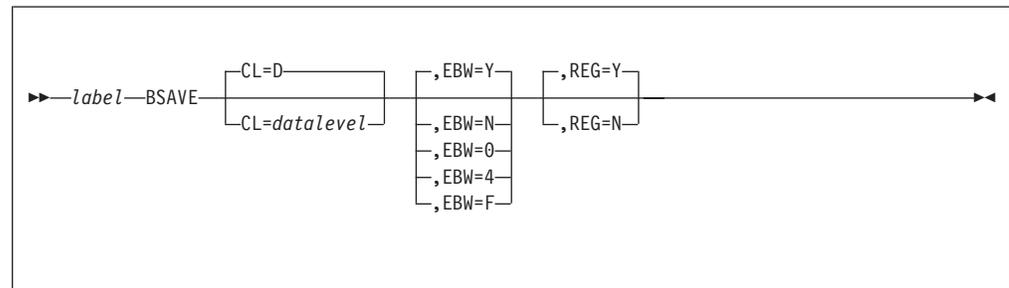
```
BRSTR
```

```
BRSTR CL=5,EBW=0,REG=N
```

## BSAVE–Recoup Register and Work Area Save

Use this system macro to save registers and sections of the entry control block (ECB) work area on the recoup stack save area.

### Format



*label*

specifies a symbolic name that can be assigned to the macro statement.

#### CL

specifies that the recoup stack save area is on an ECB data level where *datalevel* is a hexadecimal value in the range 0–F.

#### EBW

specifies one of the following:

- Y** specifies that EBW000 to EBW096 is saved in the recoup stack save area. This is the default.
- N** specifies that EBW000 to EBW103 is not saved.
- 0** specifies that EBW000 to EBW039 is saved in the recoup stack save area.
- 4** specifies that EBW040 to EBW095 is saved in the recoup stack save area.
- F** specifies that EBW000 to EBW103 is saved in the recoup stack save area.

#### REG

specifies one of the following:

- Y** specifies that general registers R0–R7 are saved in the recoup stack save area.
- N** specifies that no general registers are saved.

## Entry Requirements

- R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- R14 points to the start recoup stack block (BC0SA). R15 points to the start of the heap save area (BC2HDR). The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro is used in conjunction with BRSTR.

## BSAVE

### Examples

```
BSAVE
```

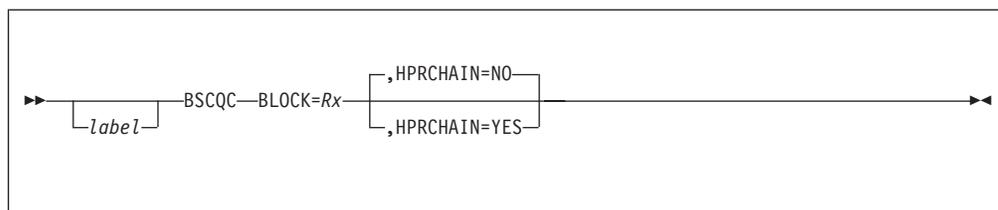
```
BSAVE CL=5,EBW=0,REG=N
```

## BSCQC—Release Core Blocks That Are Not Attached to an ECB

Use this system macro to release core blocks that are not attached to an entry control block (ECB). This macro is called from an E-type program that has the system virtual address (SVA) of the first or only core block being released.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

**BLOCK=*Rx***

A register, R0–R7, containing the SVA of the first or only core block being released.

**HPRCHAIN=YES|NO**

Specifies whether a chain of core blocks used for high-performance routing (HPR) support is being released.

### Entry Requirements

- This macro must be called from the ECB virtual memory (EVM).
- The register specified for the BLOCK parameter must contain the SVA of the first or only core block being released.
- If HPRCHAIN=YES is coded, the register specified for the BLOCK parameter must point to the first of the chained core blocks being released. These core blocks must be chained together using the NLP1FWD field contained in the core block. The end of the core block chain is indicated by NLP1FWD=0.

### Return Conditions

- Control is returned to the next sequential instruction.
- Registers remain unchanged.
- The core block pointed to by the register specified for the BLOCK parameter will be released.
- If HPRCHAIN=YES is coded, all of the chained core blocks will be released.

### Programming Considerations

None.

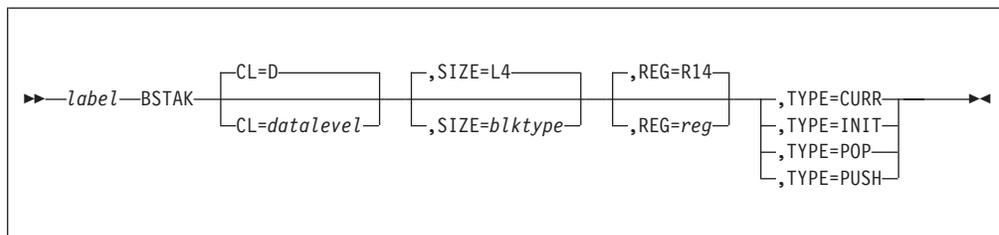
### Examples

None.

## BSTAK–Recoup Stack

Use this system macro to control the recoup stack area.

### Format



*label*

specifies a symbolic name that can be assigned to the macro statement.

**CL**

specifies that the recoup stack save area is on an entry control block (ECB) data level, where *datalevel* is a hexadecimal value in the range 0–F.

**SIZE**

specifies the size of the core block to use for a stack block, where *blktype* is one of the following valid sizes:

- L1** is for 381-byte
- L2** is for 1055-byte
- L4** is for 4095-byte

**REG**

specifies the register to which you want a pointer to the top of the recoup stack area block returned, where *reg* is one of the following valid registers: R1–R7 or R14.

**TYPE**

specifies one of the following:

**CURR**

verifies that a recoup stack area block exists on the ECB data level specified by the CL parameter and returns a pointer to the top of the recoup stack area block in the register specified by the REG parameter.

**INIT**

gets and initializes a recoup stack area block on the ECB data level specified by the CL parameter.

**POP**

pops (removes) the stack or moves data on the stack up one entry. This is used when the contents of the current stack slot are no longer needed. The first previous stack slot entry is set as the current entry.

**PUSH**

pushes the stack or moves data on the stack down one entry. This is used to save the contents of the current stack slot and make room for additional data.

### Entry Requirements

- R9 must contain the address of the ECB being processed.

- For PUSH, POP, and CURR, a valid recoup stack block must reside on the selected data level.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register specified by the REG parameter contains a pointer to the start of the recoup stack block. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro is used with BRSTR and BSAVE.
- This macro can only be used in the recoup package.

## Examples

```
BSTAK TYPE=INIT
```

```
BSTAK CL=D,TYPE=PUSH
```

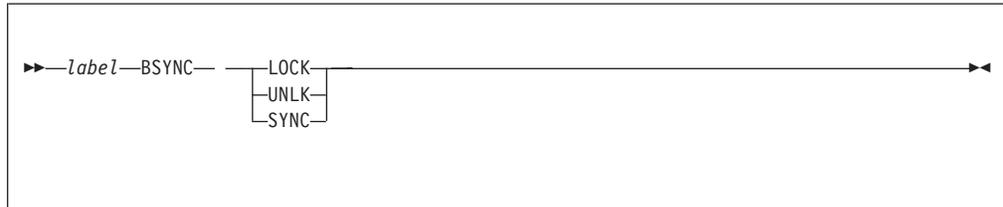
```
BSTAK CL=D,TYPE=POP
```

```
BSTAK TYPE=CURR,REG=R7
```

## BSYNC–Recopy SYNCC Facility

This system macro provides a way to synchronize important recoup control information across processors that are in a loosely coupled TPF complex.

### Format



*label*

is a symbolic name that can be assigned to the macro statement.

#### LOCK

requests exclusive use of the recoup control record or data. The core copy of the data will be refreshed.

#### UNLK

releases the exclusive use of the recoup control record. The control record must have been locked before the UNLK request.

#### SYNC

requests synchronization of recoup data in active loosely coupled processors. The control record is filed and unheld. An interprocessor communications (IPC) message is sent to the active processors to refresh their copies of the data. The control record must have been locked before the SYNC request.

### Entry Requirements

- Register 9 (R9) must contain the address of the entry control block (ECB) being processed.
- The entry must not be holding the recoup data at the time of a LOCK request.
- The entry must be holding the recoup data at the time of a UNLK or SYNC request.
- There should not be any outstanding I/O requests for this ECB.
- The LOCK condition should not be carried across an ENTxC macro.

### Return Conditions

- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.
- LOCK option:
  - Control is returned to the next sequential instruction (NSI).
  - The program status word (PSW) protect key of the ECB equals X'C'.
- SYNC option:
  - The PSW protect key of the ECB equals X'1'.
- UNLK option:
  - The PSW protect key of the ECB equals X'1'.

## Programming Considerations

None.

## Examples

The following example shows recoup data being locked before processing user code and synchronizing recoup data after the code has completed processing.

```
BSYNC      LOCK
  .....
  BEGIN PROCESSING CODE
  .
  .
  .
  END PROCESSING CODE
  .....
BYSNC      SYNC
```

## CEBIC–Change MDBF Subsystem/Subsystem User ID

Use this system macro to change the identifiers (IDs) in the ECB CE1SSU and CE1DBI fields, which allows access to the databases of other subsystems (SSs) and subsystem users (SSUs). You can use this macro and the UATBC macro together to obtain information about any SS or SSU in the TPF system. See the examples that follow. Also see *TPF General Macros* for more information about the UATBC macro.

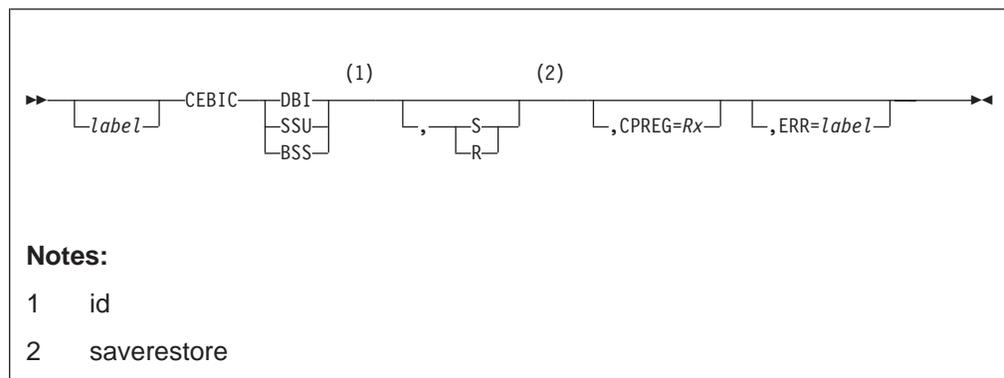
The CEBIC macro has the following functions:

- Changes the current database ID (DBI) and the current subsystem user ID (SSU ID).
- Saves the DBI and SSU ID before modification so it can return to the originating SS or SSU.
- Restores the previously saved DBI and SSU ID for return to the originating SS and SSU.

In a multiple database function (MDBF) environment, the SSU and the parent SS to which an entry control block (ECB) belongs are specified by the IDs in the ECB CE1SSU and CE1DBI fields, respectively. A program can access only the database of the SS or SSU to which its ECB belongs. That is, global and fixed file accesses are based on the SSU ID specified in the CE1SSU field. Other main storage data and random pool accesses are based on the SS ID specified in the CE1DBI field.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



*label*

A symbolic name can be assigned to the macro statement.

*id* This is a required parameter.

This specifies whether the current subsystem data base ID or the current subsystem user ID (or both) are to be changed.

### DBI

Specifies that the current subsystem data base ID and the current subsystem user ID are to be changed. For type E programs, register 14 contains the MDBF ID to which the DBI is to be changed. For C-type

programs, the CPREG parameter contains the MDBF ID to which the DBI is to be changed. The subsystem user ID is changed to the first or only SSU ID in the new subsystem.

**SSU**

Specifies that the current subsystem user ID is to be changed. If the new SSU ID does not reside within the current subsystem data base ID (CE1DBI) then the DBI is changed to the parent subsystem of the new SSU.

**BSS**

Specifies that the current subsystem data base ID is to be changed to the BSS data base ID and the current subsystem user ID is to be changed to the first or only SSU in the basic subsystem (BSS). For E-type programs, any other parameters coded on the macro call will cause assembly errors. For C-type programs, the ERR parameter is required and the CPREG parameter is optional. If coded, the CPREG parameter is ignored. Any other parameters coded on the C-type macro call will cause assembly errors.

*saverestore*

This optional parameter indicates whether the current DBI and SSU ID are to be saved or restored during this macro call.

- S** This specifies that the current DBI and SSU ID are to be saved prior to modification. This option is independent of the request type coded for the id. No mechanism is provided for saving only the DBI or only the SSU ID.
- R** Specifies that the current DBI and/or SSU ID are to be restored with the DBI and/or SSU ID last saved by a CEBIC macro call, respectively. If 'id' is DBI, both the DBI and SSU ID are restored. If 'id' is SSU, only the SSU ID is restored.

**CPREG=R<sub>x</sub>**

This parameter is required on C-type macro calls unless *id* is BSS or *saverestore* is R. (In these two cases this parameter is optional, and it is ignored if coded.) It specifies the register containing the SS or SSU ID to which the current DBI/SSU ID is to be changed. The CPREG parameter is not valid on E-type macro calls and use of this parameter by an E-type program will result in an assembly error.

**ERR=label**

This parameter is required on all C-type macro calls and is not valid on all E-type macro calls. It specifies the label to be used for error exit. Use of this parameter by an E-type program will result in an assembly error.

## Entry Requirements

- E-Type Programs:
  - R9 must contain the address of the ECB being processed.
  - R14 must contain a 2 byte MDBF ID, right-justified (1-byte complement and 1-byte ordinal number) on all calls except when *id* is BSS or *saverestore* is R. The high-order 2 bytes of the register are irrelevant.
- C-Type Programs:
  - The issuing program must be in supervisor state.
  - R9 must contain an ECB address.

## CEBIC

- The register specified in CPREG must contain a 2-byte MDBF ID, right-justified (1-byte complement and 1-byte ordinal number) on all calls except when *id* is BSS or *saverestore* is R. The high-order 2 bytes of the specified register are irrelevant.

## Return Conditions

- Normal Return Condition:
  - Control is returned to the next sequential instruction following the macro call.
  - R0 through R9 are unchanged.
  - The contents of general registers 14 and 15 are unknown.
  - The ECB fields CE1DBI and CE1SSU are changed to the requested subsystem and subsystem user as defined in “Format” on page 122.
  - If the original SSU ID is changed, the global base addresses CE1GLA and CE1GLY in the ECB are modified to reflect the new SSU.
  - If the DBI is changed, its active ECB counter (CA2NEB) is decreased by 1 and the active ECB counter of the new SS is increased by 1.
- Error Return:
  - If an error is encountered while servicing a CEBIC macro issued from a C-type segment, a branch is taken to the label specified on the ERR parameter.  
No fields are modified.
  - If an error is encountered while servicing a CEBIC macro issued from an E-type segment, a system error is issued and the ECB is exited.  
No fields are modified.  
The following lists error conditions that produce system errors:
    - MDBF ID is not valid  
CE1DBI or CE1SSU upon input to the CEBIC service routine failed the integrity check (that is, the sum of the ordinal number and complement byte was not equal to X'FF').  
The DBI or SSU ID supplied in R14 or the specified CPREG failed the integrity check.  
The DBI or SSU ID to be restored failed the integrity check.
    - CEBIC CP register save area in use  
The CEBIC macro was issued from a C-type segment when the CEBIC CP register save area was in use. This condition should not occur and causes a catastrophic error. Only one CEBIC macro issued from a C-type segment can be serviced at a time.
    - MDBF ID out of range  
CE1DBI or CE1SSU upon input to the CEBIC service routine references a SS/SSU, which was not included in the last IPL.  
The DBI or SSU ID provided in R14 or the specified CPREG references a DBI or SSU ID that was not included in the last IPL.  
The DBI or SSU ID to be restored references a DBI or SSU ID that was not included in the last IPL.
    - Subsystem inactive  
CE1DBI upon input to the CEBIC service routine references a subsystem that is indicated as inactive in CTKM.  
The DBI provided in R14 or the specified CPREG references a subsystem that is indicated as inactive in CTKM.

The parent subsystem of the SSU provided in R14 or the specified CPREG references a subsystem that is indicated as inactive in CTKM.

The DBI to be restored references a subsystem that is indicated as inactive in CTKM.

- Subsystem User ID does not reside within Subsystem

The current SSU ID does not reside within the current DBI.

On a DBI,R request, the SSU ID to be restored does not reside within the DBI to be restored.

On a SSU,R request, the SSU ID to be restored does not reside within the current DBI (CE1DBI).

## Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called from either E-type or C-type programs.
  - The macro expands to an SVC for E-type programs.
  - The macro expands to a BASSM for C-type programs.
- This macro does not alter the program base identifier (CE1PBI).
- The CEBIC macro expansion is independent of the operating environment. However, the macro service routine provided for a system generated with a MDBF environment is different from that generated for a system without such an environment.
- Data base destruction can occur if care is not exercised in the use of this macro. If, for example, a record was found with an SSU ID and later filed after the SSU ID was changed, data base destruction may occur.
- The SAVE option is independent of the request type. That is, on both DBI and SSU requests, the SAVE option causes both the current DBI and the current SSU to be saved before modification.
- There is no mechanism for restoring only the DBI. On a DBI request, the restore option restores both the DBI and SSU. On an SSU request, the restore option restores only the SSU ID.
- Valid registers for the CPREG parameter are R0 through R8.
- To avoid the condition that raises a system error the CEBIC macro should not be issued from a CP program that is interruptible.

## Examples

The following example from an E-type segment is based on a MDBF system configured as follows.

SS Name	SS MDBF ID	SSU Name	SSU Mass ID
BSS	FF00	SSU1 SSU2	FF00 FE01
SS1	FE01	SSU3 SSU4 SSU5	FD02 FC03 FB04
SS2	FD02	SSU6	FA05

Assume that upon entry to the code sequence CE1DBI FF00 (BSS) and CE1SSU FE01 (SSU2), and that these fields must be unchanged upon exit. The example illustrates one method of how the CEBIC macro can be used with the UATBC macro to obtain the names of all subsystem users in the next active subsystem.

## CEBIC

	MSOUT REG=R2	06-SSUT ADDRESSABILITY
	LEBIC DBI,R2,CHECK=NO	GET CURRENT SSORDINAL NUMBER
		(See Note 1)
NEXT	LA R2,1(,R2)	BUMP TO NEXT SS
		(See Note 2)
	UATBC IDLOC=(R,SSI,R2),	CHECK VALIDITY OF NEXT SS
	EXCD=SSTOOBIG,	..TOO BIG, WE ARE FINISHED
	INVLID=SSNEG,	..SS IS INVALID
	NOTAVL=SSINACT,	..SS IS INACTIVE
	COUNT=R3	..NUMBER OF SSUs IN SS
		(See Note 3)
	LH R14,MU0DPI	SS ID OF NEXT SS
		(See Note 4)
	CEBIC DBI,S	SAVE CURRENT SS &
		..BUMP ENTRY TO NEXT SS
		(See Note 5)
LOOP	UATBC IDLOC=(E,SSU,R2),	GET THE SSUT ADDRESS FOR ID
	NOTAVL=SSUDORM	..IN CE1SSU
		(See Note 6)
	WTOPC TEXT='SSU NAME IS...',	
	SUB=(CHARA,MU0NAM)	
SSUDORM	DS 0H	
	LA R2,MU0LEN(,R2)	BUMP TO NEXT SSUT SLOT
		(See Note 7)
	BCT R3,LOOP	LOOP THRU ALL SSUs FOR THIS
		SS (See Note 8)
	CEBIC DBI,R	RESTORE DBI & SSU ID TO
		ORIGINAL (See Note 9)
SSTOOBIG	DS 0H	THIS SS IS LAST, NEXT IS
		TOO BIG
SSNEG	DS 0H	INPUT SS IS INVALID
	EXITC	
SSINACT	DS 0H	SS IS INACTIVE;
	B NEXT	...TRY NEXT ONE

### Notes:

1. The following shows the input and the output for Note 1.

**Input:** CE1DBI FF00

**Output:** R2 00 (high-order 3 bytes irrelevant)

2. The following shows the input and the output for Note 2.

**Input:** R2 00 (first time through loop)

**Output:** R2 01

3. The following shows the input and the output for Note 3.

**Input:** R2 01 R3 irrelevant

**Output:** R3 3 (number of SSUs in SS1)

4. The following shows the input and the output for Note 4.

**Output:** R14 FE01

5. The following shows the input and the output for Note 5.

**Input:**

CE1SDBI irrelevant  
 CE1DBI FF00  
 CE1SSU FE01  
 R14 FE01

**Output:**

CE1SDBI FF00 FE01  
 CE1DBI FE01  
 CE1SSU FD02

6. The following shows the input and the output for Note 6.

**Input:**

R2 irrelevant  
 CE1SSU FD02

**Output:** R2 SSUT slot address of SSU3

7. The following shows the input and the output for Note 7.

**Input:** First time through loop R2 SSU slot address of SSU3

**Output:** First time through loop R2 SSUT slot address of SSU4

8. The following shows the input for Note 8.

**Input:** First time through loop R3 3

9. The following shows the input and the output for Note 9.

**Input:**

CE1DBI FE01  
 CE1SSU FB04  
 CE1SDBI FF00 FE01

**Output:**

CE1DBI FF00  
 CE1SSU FE01

---

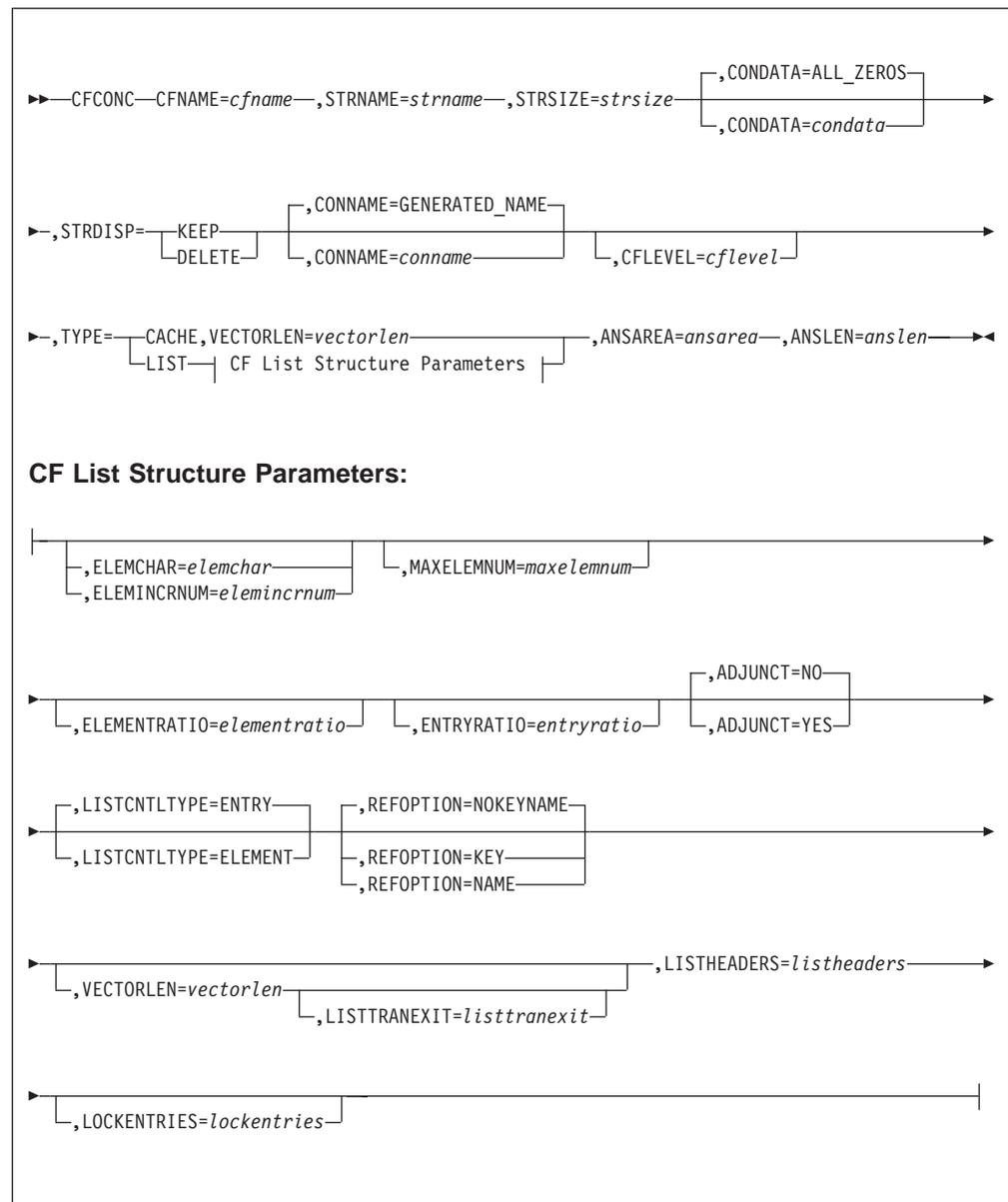
## CFCONC—Connect to a Coupling Facility List or Cache Structure

Use this system macro to allocate and connect to a coupling facility (CF) list or cache structure on a CF, or to connect to a CF list or cache structure that is allocated already. A *CF list or cache structure* is a named piece of storage on a CF.

The first user to connect to a CF list or cache structure allocates the structure on a CF and defines the structure attributes, including the type of CF structure (list or cache). Other users can connect to the CF structure by name, but cannot change the structure attributes of the CF structure as long as it remains allocated. All connectors, whether the first or subsequent, are informed of the structure attributes through the CFCONC answer area, which is mapped by the ICFCAA DSECT. Users are responsible for checking the structure attributes to verify that they are acceptable.

See *TPF Database Reference* for more information about connecting to a CF list or cache structure.

## Format



## Parameters Common to Both Structure Types

**CFNAME=cfname**

Specifies the CF name, which is needed to allocate the CF list or cache structure. The CF name must be a 5- to 8-character alphanumeric name, and the first character must be an alphabetic character.

To code, specify the name or address of the 8-character field that contains the CF name in which the CF list or cache structure is to be allocated.

**STRNAME=strname**

Specifies the name of the CF list or cache structure to which you want to connect. The name you specify must meet the following requirements:

- The name must begin with an uppercase alphabetic character. TPF-defined structure names begin with I or TPF.

## CFCONC

- The name must be 16 characters long and padded on the right with blanks if necessary.
- The name can contain numeric characters, uppercase alphabetic characters, and underscores.

To code, specify the name or address of the 16-character input field that contains the CF list or cache structure name.

### **STRSIZE=***strsize*

Specifies the size of the CF list or cache structure in 4-KB blocks. This is a required parameter.

**Note:** If there are limited CF resources available, the CF list or cache structure can be allocated with less space than you requested. The actual size of the CF list or cache structure allocated is returned in the ICFCAASTRSIZE field of the CFCONC answer area.

To code, specify the name or address of the fullword input field that contains the number of 4-KB blocks that make up the CF list or cache structure.

### **CONDATA**

Specifies the connect data to pass to your exit routines, where:

#### ALL\_ZEROS

Sets the field to zeros.

#### *condata*

The 8 bytes of connection data.

The connection data is passed to all of the connector exits and is for your use only; the TPF system does not use this information. A possible use for the connection data is as a pointer to a control block that represents the connector.

To code, specify the name or address of the 8-character input field that contains the connect data.

### **STRDISP**

Specifies the disposition or persistence attribute of the CF list or cache structure when all connections are released.

The structure disposition determines whether the CF list or cache structure remains allocated or not when there are no connections to the CF list or cache structure.

Specify one of the following:

#### **KEEP**

Indicates that when there are no connections to the CF list or cache structure, it remains allocated. For example, if the data structure must be kept permanently on a CF, specify a structure disposition of KEEP. A CF list or cache structure that remains allocated when there are no connections is called a *persistent structure*.

#### **DELETE**

Indicates that the CF list or cache structure becomes unallocated when there are no connections to the CF list or cache structure.

### **CONNAME**

Specifies a connection name that identifies your connection to the CF list or cache structure where:

**GENERATED\_NAME**

Indicates a unique connection name to be generated by the TPF system.

*conname*

Indicates a unique name for each connection to a CF list or cache structure and meets the following requirements:

- The name must begin with an uppercase alphabetic character.
- The name must be 16 characters long and padded on the right with blanks if necessary.
- The name can contain numeric characters, uppercase alphabetic characters, and underscores.

To code, specify the name or address of a 16-character field that contains the unique connection name to identify your connection to the CF list or cache structure.

**CFLEVEL=*cflevel***

Specifies the CF level of the CF. The connector requires the CF list or cache structure to be allocated in at least this specified CF level. If you specified a CF level that is higher than that supported by the TPF system on which the requester runs, the connection is not successful.

The ICFCAATPFMAXCFLEV field of the CFCONC answer area contains the maximum CF level supported by the TPF system. The ICFCAACFLEVEL field contains the actual CF level of the CF in which the CF list or cache structure was allocated.

To change to a different CF level, you must disconnect from the CF list or cache structure and then connect to it again, specifying a different CF level. See “CFDISC—Disconnect from a Coupling Facility List or Cache Structure” on page 143 for more information about disconnecting from a CF list or cache structure.

To code, specify the name or address of the fullword input field that contains the CF level required.

**TYPE**

Identifies the type of CF structure in the CF to which you want to connect.

Specify one of the following:

**LIST**

Indicates that the structure is a CF list structure.

**CACHE**

Indicates that the structure is a CF cache structure.

**ANSAREA=*ansarea***

Specifies the address of the CFCONC answer area. When the CFCONC macro has been completed, it returns information to the caller in the CFCONC answer area. The requester can use the ICFCAA DSECT to map the CFCONC answer area. The CFCONC answer area must begin on a doubleword boundary and storage for this parameter must be in the same 4 K page.

To code, specify the name or address of the CFCONC answer area.

**ANSLEN=*anslen***

Specifies the length of the CFCONC answer area. The length should be long enough to accommodate the ICFCAA DSECT mapping of the CFCONC answer area.

## CFCONC

To code, specify the name or address of a fullword input field that contains the length of the CFCONC answer area.

### Parameter for CF Cache Structures (TYPE=CACHE)

#### **VECTORLEN**=*vectorlen*

Specifies the number of cache buffers in the local storage of the requester that require concurrent registration. The requester uses the vector length to map each local cache buffer to a named data entry in the CF cache structure. The value you specify must be a multiple of 32 or the TPF system rounds up the value to a multiple of 32.

To code, specify the name or address of the fullword field that contains the number of cache buffers in the local storage of the requester.

### Parameters for CF List Structures (TYPE=LIST)

#### **ELEMCHAR**=*elemchar*

Specifies the value of the element characteristic, which is used to determine the CF list structure element size. This size is calculated with the formula  $256 \times (2^{**ELEMCHAR})$ , where ELEMCHAR is used as the power of 2. For example, if the value referenced by the ELEMCHAR parameter is 0, the size of each CF list structure element is 256 bytes.

Valid values for the ELEMCHAR parameter range from 0 to a maximum determined by the CF model limitation on the CF list structure element size.

**Note:** The CF list structure element size determines the size of the data entry, which is the data written to and read from the CF list structure. A data entry can be up to 16 times the data element size as indicated by the element characteristic.

Specify either the ELEMCHAR or ELEMINCRNUM parameter to define the CF list structure element size. If neither parameter is specified, the CF list structure is allocated as if the value of the ELEMCHAR parameter was specified as 0.

To code, specify the name or address of the 1-byte field that contains the value of the element characteristic.

#### **ELEMINCRNUM**=*elemincrnnum*

Specifies the element increment number that is used to determine the CF list structure element size. This size is calculated with the formula  $256 \times ELEMINCRNUM$ . For example, if the value referenced by the ELEMCHAR parameter is 1, the size of each CF list structure element is 256 bytes.

The valid values for ELEMINCRNUM range from 1 to a maximum determined by the CF model limitation on the size of the CF list structure element. The value you specify must be a power of 2.

**Note:** The CF list structure element size determines the size of the data entry, which is the data written to and read from the CF list structure. A data entry can be up to 16 times the data element size as indicated by the element increment number.

Specify either the ELEMCHAR or the ELEMINCRNUM parameter to define the CF list structure element size. If neither parameter is specified, the CF list structure is allocated as if the value of the ELEMCHAR parameter was specified as 0.

To code, specify the name or address of the 1-byte field that contains the value of the element increment number.

**MAXELEMNUM**=*maxelemnum*

Specifies a value that determines the maximum number of data elements for each list entry in the CF list structure. The value you specify can be from 1 to 16 bytes. If you do not specify a value, a value of 16 is provided as the default. The maximum list entry size, in bytes, equals the value specified for the MAXELEMNUM parameter multiplied by the CF list structure element size obtained from the value you specified for either the ELEMCHAR or ELEMINCRNUM parameter. For example, if the value referenced by the ELEMCHAR parameter is 0 and the value referenced by the MAXELEMNUM parameter is 1, the maximum list entry size is 256 bytes. If the value referenced by the ELEMCHAR parameter is 4 and the value referenced by the MAXELEMNUM parameter is 16, the maximum list entry size is 65 536.

To ensure that the CF can assign all possible data elements allocated in a CF list structure to a list entry, the value specified for this parameter must be greater than or equal to the value specified for the ELEMENTRATIO parameter divided by the value specified for the ENTRYRATIO parameter. The MAXELEMNUM parameter is ignored if the value referenced by the ELEMENTRATIO parameter is 0.

If this parameter is not specified, the CF list structure is allocated as if the value of the MAXELEMNUM parameter was specified as 16.

To code, specify the name or address of the 2-byte field that contains a value that determines the maximum number of data elements for each list entry in the CF list structure.

**ELEMENTRATIO**=*elementratio*

Specifies a value to express the data element portion of the entry-to-element ratio of the CF list structure. If a value of 0 is specified for the ELEMENTRATIO parameter, the CF list structure is allocated without data elements. If this parameter is not specified, the CF list structure is allocated as if the value of the ELEMENTRATIO parameter was specified as 1.

To code, specify the name or address of the 2-byte field that contains a value to express the data element portion of the entry-to-element ratio of the CF list structure.

**ENTRYRATIO**=*entryratio*

Specifies a value to express the list entry portion of the entry-to-element ratio of the CF list structure.

To code, specify the name or address of the 2-byte field that contains a value to express the list entry portion of the entry-to-element ratio.

**ADJUNCT**

Specifies whether adjunct data areas are associated with each list entry in the CF list structure. Each adjunct data area can contain 64 bytes of user-defined data such as information about the status of the data entry or time stamp. You can specify one of the following:

**NO**

Indicates that there are no adjunct data areas specified for the CF list structure.

**YES**

Indicates that each list element in the CF list structure has an associated adjunct data area of 64 bytes.

**LISTCNTLTYPE**

Specifies whether the CF maintains and tracks list limits based on the number of list entries or the number of data elements for each list.

Specify one of the following:

**ENTRY**

Specifies that the list limits are identified and tracked as limits on the number of entries that may reside on the list. For example, if your main concern is to limit the number of entries that might build up on a list, limit the number of list entries for each list.

**ELEMENT**

Specifies that the list limits are identified and tracked as limits on the total number of data elements that may be associated with entries on the list. For example, if your main concern is to prevent the entries on a given list from consuming too much of the storage in a CF list structure, limit the number of data elements on a list.

**REFOPTION**

Specifies how to reference list entries in the CF list structure. Specify one of the following:

**NOKEYNAME**

Indicates that the list entry is located by the list entry identifier (LEID). The LEID is assigned for each list entry that is in use in the CF list structure. Keys or names are not used to reference list entries.

**KEY**

Indicates that the list entry is located by a key value. When creating the list entry, you can assign a key value to one or more list entries. Keyed entries allow users to maintain list entries in a keyed order. For example, if the list entries represent units of work ordered by priority, you could choose keyed entries.

**NAME**

Indicates that the list entry is located by a unique name. When creating the list entry, you can assign a unique name to each list entry. Named entries allow users to reference list entries by a user-defined name. If the list entries represent customer records in a particular order, you could choose named entries.

**VECTORLEN=*vectorlen***

Specifies the number of list notification vector entries in the vector that this connection will be monitoring for list transitions. A list transition occurs when an empty list in the CF list structure becomes nonempty (changes from an empty state to a nonempty state). This value is rounded up to a multiple of 32.

To code, specify the name or address of the fullword field that contains the number of list notification vector entries in the vector.

**LISTTRANEXIT=*listtranexit***

Identifies the address of the list transition exit routine for the requester. The list transition exit routine notifies you when one or more lists that you are monitoring changed from an empty state to a nonempty state.

To code, specify the name or address of the list transition exit for the requester.

**LISTHEADERS=*listheaders***

Specifies the number of lists to be allocated in the CF list structure. This number must be greater than 0.

To code, specify the name or address of the fullword input field that contains the number of list headers you want allocated.

**LOCKENTRIES=***lockentries*

Specifies the number of lock entries for the CF list structure. If this value is not a power of 2, it is rounded up to the nearest power of 2. If you do not specify a value or the requester specifies 0, the CF does not support serialization for the allocated CF list structure.

To code, specify the name or address of the fullword field that contains the number of lock entries for the CF list structure.

## Entry Requirements

- The CFCONC macro must be issued from an E-type program.
- The maximum number of connections to a CF list or cache structure on a CF is set by the CF control code. This limit depends on the CF model. CFs at level 6 allow up to 32 connections to a CF list or cache structure.
- The combination of the MAXELEMNUM parameter and either the ELEMCHAR or ELEMINCRNUM parameter must define a data entry size less than or equal to 64 K.

## Return Conditions

When control returns to the caller of the CFCONC macro, the general-purpose registers (GPRs) contain the following:

Register	Contents
15	Return code in the high-order 2 bytes and a reason code in the low-order 2 bytes.

All other registers remain unchanged. Table 1 shows the hexadecimal return code, reason code, and equate symbol associated with each reason code. The ICFEQ DSECT provides equate symbols for the return and reason codes. The following are the equate symbols associated with each hexadecimal return code:

Return Code	Equate Symbol
0000	ICFRRCOK
0004	ICFRRCWARNING
0008	ICFRRCPARMERROR
000C	ICFRRCENVEERROR
0010	ICFRRCCOMPONENT.

Table 1. Return and Reason Codes for the CFCONC Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0000	None	<p><b>Equate Symbol:</b> None.</p> <p><b>Meaning:</b> The function is completed successfully and the TPF system returns data to the CFCONC answer area. Use the ICFCAA DSECT to map the CFCONC answer area.</p> <p><b>Action:</b> None. It is the responsibility of the user to verify that the structure attributes, as recorded in the CFCONC answer area, are acceptable.</p>

## CFCONC

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0008	0801	<p><b>Equate Symbol:</b> ICFRRCBADPARMLIST</p> <p><b>Meaning:</b> A program error occurred because the address of the CFCONC parameter list is zero.</p> <p><b>Action:</b> Verify that the address is not corrupted.</p>
0008	0804	<p><b>Equate Symbol:</b> ICFRRCBADVERSIONNUM</p> <p><b>Meaning:</b> A program error occurred because the version number found in the CFCONC parameter list is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Verify that your program did not overlay the parameter list storage.</li> <li>2. Verify that your program was assembled with the correct macro library for the release of the TPF system on which your program is running.</li> </ol>
0008	080D	<p><b>Equate Symbol:</b> ICFRRCAREATOOSMALL</p> <p><b>Meaning:</b> A program error occurred because the CFCONC answer area is too small as indicated by the length of the CFCONC answer area specified on the ANSLEN parameter.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Ensure that the value specified on the ANSLEN parameter correctly reflects the size of the CFCONC answer area. The size of the CFCONC answer area is specified on the ANSLEN parameter.</li> <li>2. Ensure that the length of the CFCONC answer area is large enough to contain the data returned.</li> </ol>
0008	080E	<p><b>Equate Symbol:</b> ICFRRCBADAREA</p> <p><b>Meaning:</b> A program error occurred because a pointer to the CFCONC answer area is null.</p> <p><b>Action:</b> Provide a valid pointer in your program.</p>
0008	081B	<p><b>Equate Symbol:</b> ICFRRCNOLENTRIES</p> <p><b>Meaning:</b> A program error occurred because the number of lock entries specified on the LOCKENTRIES parameter is 0.</p> <p><b>Action:</b> If the lock entries are in use, ensure that the value specified on the LOCKENTRIES parameter is greater than 0.</p>
0008	081C	<p><b>Equate Symbol:</b> ICFRRCNOLISTHDRS</p> <p><b>Meaning:</b> A program error occurred because the number of list headers specified on the LISTHEADERS parameter is 0.</p> <p><b>Action:</b> Ensure that the value specified on the LISTHEADERS parameter is greater than 0.</p>

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0008	081F	<p><b>Equate Symbol:</b> ICFRRCCONNAME</p> <p><b>Meaning:</b> A program error occurred because the connection name specified for the CONNAME parameter matches the connection name of another active connection to the same CF structure. The connection name you specify must be unique for each connection to a CF structure.</p> <p><b>Action:</b> Ensure that the connection name specified for the CONNAME parameter is unique and not already connected to the CF structure or allow the TPF system to generate a unique connection name.</p>
0008	0820	<p><b>Equate Symbol:</b> ICFRRCSTRTYPE</p> <p><b>Meaning:</b> A program error occurred because the CF structure type specified does not match the CF structure type allocated previously. When you connect to an allocated CF structure, you cannot change the structure type attribute. The request fails.</p> <p><b>Action:</b> Specify a value for the TYPE parameter that matches the CF structure type specified for the original CF structure.</p>
0008	0821	<p><b>Equate Symbol:</b> ICFRRCSTRSERIAL</p> <p><b>Meaning:</b> A program error occurred because the serialization attribute for a CF list structure, which is specified on the LOCKENTRIES parameter, does not match the previously allocated CF list structure. When you connect to an allocated CF list structure, you cannot change the structure attributes.</p> <p><b>Action:</b> Specify a value on the LOCKENTRIES parameter that matches the value originally specified for the CF list structure.</p>
0008	0823	<p><b>Equate Symbol:</b> ICFRRCCONNAMEERR</p> <p><b>Meaning:</b> A program error occurred because the connection name specified on the CONNAME parameter is not valid.</p> <p><b>Action:</b> Verify that the connection name specified is valid.</p>
0008	0824	<p><b>Equate Symbol:</b> ICFRRCFNAMEERR</p> <p><b>Meaning:</b> A program error occurred because the CF name specified on the CFNAME parameter is not valid.</p> <p><b>Action:</b> Verify that the CF name specified is valid.</p>
0008	0825	<p><b>Equate Symbol:</b> ICFRRCSTRNAMEERR</p> <p><b>Meaning:</b> A program error occurred because the structure name specified on the STRNAME parameter is not valid.</p> <p><b>Action:</b> Verify that the structure name specified is valid.</p>

## CFCONC

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0008	085A	<p><b>Equate Symbol:</b> ICFRRCINVALIDCACHEPARM</p> <p><b>Meaning:</b> A parameter that is only valid for a CF list structure was specified for a CF cache structure. See <i>Parameter for CF Cache Structures (TYPE=CACHE)</i> for information about valid CF cache parameters.</p> <p><b>Action:</b> Verify that the parameters specified are valid for a CF cache structure.</p>
0008	085F	<p><b>Equate Symbol:</b> ICFRRCINVALIDVECTORLEN</p> <p><b>Meaning:</b> A program error occurred because the value specified for the VECTORLEN parameter for a CF cache structure was equal to zero. The request fails.</p> <p><b>Action:</b> Specify a nonzero value for the VECTORLEN parameter for a CF cache structure.</p>
0008	0861	<p><b>Equate Symbol:</b> ICFRRCENTRYRATIO</p> <p><b>Meaning:</b> A program error occurred because the values specified on the ELEMENTRATIO and ENTRYRATIO parameters are not valid. When the value specified for the ELEMENTRATIO parameter is greater than 0, the value specified for the ENTRYRATIO parameter must also be greater than 0.</p> <p><b>Action:</b> Ensure you specify a value that is greater than 0 for the ENTRYRATIO parameter.</p>
0008	0862	<p><b>Equate Symbol:</b> ICFRRCMAXELEMNUM</p> <p><b>Meaning:</b> A program error occurred because the values specified for the ELEMENTRATIO and MAXELEMNUM parameters are not valid. When allocating a CF list structure, if the value specified for the ELEMENTRATIO parameter is not 0, the value specified for the MAXELEMNUM parameter must be greater than or equal to that value, divided by the value specified for the ENTRYRATIO parameter.</p> <p><b>Action:</b> Ensure you modify the value specified for the MAXELEMNUM parameter so that it meets these requirements.</p>
0008	0864	<p><b>Equate Symbol:</b> ICFRRCCELEMINCRNUMELEMCHAR</p> <p><b>Meaning:</b> A program error occurred because both the ELEMCHAR and ELEMINCRNUM parameters were specified. Only one of these parameters can be specified.</p> <p><b>Action:</b> Issue the CFCONC macro again, specifying either the ELEMCHAR or ELEMINCRNUM parameters.</p>
0008	086B	<p><b>Equate Symbol:</b> ICFRRCCELEMINCRNUM</p> <p><b>Meaning:</b> A program error occurred because the value specified on the ELEMINCRNUM parameter is not valid.</p> <p><b>Action:</b> Ensure you specify a value greater than 0 and a power of 2.</p>

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0008	0871	<p><b>Equate Symbol:</b> ICFRRCMAXELEMNUMELEMCHAR</p> <p><b>Meaning:</b> A program error occurred because the values specified on the MAXELEMNUM parameter and either the ELEMCHAR or ELEMINCRNUM parameter will result in a list entry having a maximum data entry size greater than the 64-KB limit.</p> <p><b>Action:</b> Ensure you change the values specified on the MAXELEMNUM and ELEMCHAR parameters, as appropriate, to meet the 64-KB limit.</p>
0008	0881	<p><b>Equate Symbol:</b> ICFRRCBADCFLEVEL</p> <p><b>Meaning:</b> A program error occurred because you tried to use a function that is not supported by the CF level specified on the CFLEVEL parameter. The connector requires that the CF list structure be allocated in at least this specified CF level. If you specify a CF level that is higher than that supported by the TPF system on which the requester runs, the connection is not successful.</p> <p><b>Action:</b> Issue the CFCONC macro again specifying, a value for the CFLEVEL parameter that supports the functions you want to use.</p>
0008	08F2	<p><b>Equate Symbol:</b> ICFRRCRESTARTINCOMPLETE</p> <p><b>Meaning:</b> A program error occurred because the TPF system has not yet completed CF restart.</p> <p><b>Action:</b> Ensure that you wait until CF restart is completed.</p>
0008	08F3	<p><b>Equate Symbol:</b> ICFRRCCONAUTHERR</p> <p><b>Meaning:</b> An attempt was made to connect to a CF locking structure, but another processor is connected to the structure with the processor ID of this processor.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Ensure that no other processor performs an initial program load (IPL) with the processor ID of this processor.</li> <li>2. Enter <b>ZPSMS PR FORCE DEACT</b> <i>processor ID</i> to ensure that any processor that was previously IPLed with the processor ID of this processor has been deactivated.</li> </ol>
000C	0C02	<p><b>Equate Symbol:</b> ICFRRCNOMORECONNS</p> <p><b>Meaning:</b> This is an environmental error that occurred because the CF structure already has the maximum number of allowed connections.</p> <p><b>Action:</b> Try your request again at a later time.</p>
000C	0C03	<p><b>Equate Symbol:</b> ICFRRCNOMORESTRUCTS</p> <p><b>Meaning:</b> This is an environmental error that occurred because the CF cannot allocate a spare CF structure identifier (ID).</p> <p><b>Action:</b> Try your request again at a later time.</p>

## CFCONC

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
000C	0C06	<p><b>Equate Symbol:</b> ICFRRCNOCONN</p> <p><b>Meaning:</b> This is an environmental error that occurred because the TPF system has not established a connection to the CF that contains the specified CF list structure. Issuing CF commands such as VARY PATH OFFLINE or CONFIG CHP OFFLINE and hardware errors such as CF failures or path failures, may prevent this connection from being established.</p> <p><b>Action:</b> Determine why the connectivity was not established and try your request again.</p>
000C	0C11	<p><b>Equate Symbol:</b> ICFRRCDEFINE</p> <p><b>Meaning:</b> This is an environmental error that occurred because the local list notification vector could not be defined.</p> <p><b>Action:</b> Try your request again at a later time.</p>
000C	0C24	<p><b>Equate Symbol:</b> ICFRRCINVSTRSIZE</p> <p><b>Meaning:</b> This is an environmental error that occurred because the size specified for the CF structure was not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Specify a valid value for the size of the CF structure.</li> <li>2. Try your request again, specifying a valid CF structure size for the STRSIZE parameter.</li> </ol>
000C	0C25	<p><b>Equate Symbol:</b> ICFRRCSTRFAILURE</p> <p><b>Meaning:</b> This is an environmental error that occurred because of a CF structure failure.</p> <p><b>Action:</b> See your system programmer for more information.</p>
000C	0C26	<p><b>Equate Symbol:</b> ICFRRCREALLOCERROR</p> <p><b>Meaning:</b> This is an environmental error that occurred because a second allocation to change the CF structure name to the requested name was not successful.</p> <p><b>Action:</b> See your system programmer for more information.</p>
000C	0C53	<p><b>Equate Symbol:</b> ICFRRCLEVEL</p> <p><b>Meaning:</b> The CF level you specified is greater than the current CF level supported.</p> <p><b>Action:</b> See your system administrator for more information.</p>
000C	0CF0	<p><b>Equate Symbol:</b> ICFRRCAUTHLOCKERROR</p> <p><b>Meaning:</b> The connect request was rejected because the CF lock was not obtained because of an error that occurred. This error occurs when the CF fails to respond to CF commands.</p> <p><b>Action:</b> None.</p>

Table 1. Return and Reason Codes for the CFCONC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
000C	0CF2	<p><b>Equate Symbol:</b> ICFRRCSYNCERROR</p> <p><b>Meaning:</b> Your request to connect to a CF list structure on the CF was not successful because the time stamp in the structure user controls did not match the time stamp found in the coupling facility structure block (CFSB). The CFSB is deleted.</p> <p><b>Action:</b> Try to connect to the CF list structure again.</p>
000C	0CF3	<p><b>Equate Symbol:</b> ICFRRCFNOTADDED</p> <p><b>Meaning:</b> A program error occurred because the specified CF was not added to the processor configuration.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Enter the ZMCFT ADD command to add the specified CF to the processor configuration.</li> <li>2. Issue your request again.</li> </ol> <p>See <i>TPF Operations</i> for more information about the ZMCFT ADD command.</p>
000C	0CF4	<p><b>Equate Symbol:</b> ICFRRCFNOTACTIVE</p> <p><b>Meaning:</b> A program error occurred because the specified CF was not added to the processor configuration.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Enter the ZMCFT ADD command to add the specified CF to the processor configuration.</li> <li>2. Issue your request again.</li> </ol> <p>See <i>TPF Operations</i> for more information about the ZMCFT ADD command.</p>
000C	0CF5	<p><b>Equate Symbol:</b> ICFRRCFCSBFILEERROR</p> <p><b>Meaning:</b> The connect request was not successful because the TPF system could not create the coupling facility structure block (CFSB) or update it on file. This error may be accompanied by additional error messages that provide more information about the error.</p> <p><b>Action:</b> Determine the cause of the error and try your connect request again.</p>
0010	1000	<p><b>Equate Symbol:</b> ICFRRCOMPERROR</p> <p><b>Meaning:</b> A component error occurred because a CF command was not successful.</p> <p><b>Action:</b> See your system programmer for more information.</p>

## Programming Considerations

- The ICFAA data macro generates the ICFAA DSECT, which maps the area identified by the ANSAREA parameter.
- Applications should use the newCache function with the processor shared (Cache\_ProcS) parameter specified to create a CF cache structure rather than

## CFCONC

using the CFCONC macro. See *TPF C/C++ Language Support User's Guide* for more information about the newCache function.

## Examples

None.

## CFDISC—Disconnect from a Coupling Facility List or Cache Structure

Use this system macro to disconnect from a coupling facility (CF) list or cache structure on a CF when you no longer require access to it. See *TPF Database Reference* for more information about disconnecting from a CF list or cache structure.

### Format

```
▶▶—CFDISC—CONTOKEN=contoken————▶▶
```

#### **CONTOKEN=*contoken***

Specifies the connect token that was returned by the CFCONC macro. The *connect token* uniquely identifies the connection to a CF list or cache structure in your processor configuration. See “CFCONC—Connect to a Coupling Facility List or Cache Structure” on page 128 for more information about the CFCONC macro.

To code, specify the name or address of the 16-character field that contains the connect token.

### Entry Requirements

The CFDISC macro must be issued from an E-type program.

### Return Conditions

When the CFDISC macro returns control to your program you will receive a return code in the high-order 2 bytes of register 15 (R15) and a reason code in the low-order 2 bytes of R15. All other registers remain unchanged.

Table 2 on page 144 shows the hexadecimal return code, reason code, and equate symbol associated with each reason code. The ICFEQ DSECT provides equate symbols for the return and reason codes. The following are the equate symbols associated with each hexadecimal return code:

<b>Return Code</b>	<b>Equate Symbol</b>
0	ICFRRCOK
4	ICFRRCWARNING
8	ICFRRCPARMERROR
C	ICFRRCENVEERROR
10	ICFRRCCOMPONENT.

## CFDISC

Table 2. Return and Reason Codes for the CFDISC Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
0000	0000	<p><b>Equate Symbol:</b> None.</p> <p><b>Meaning:</b> The request to disconnect from a CF structure is successful. The connect token is no longer valid and will be rejected for any subsequent disconnect requests.</p> <p><b>Action:</b> None.</p>
0008	0801	<p><b>Equate Symbol:</b> ICFRRCBADPARMLIST</p> <p><b>Meaning:</b> A program error occurred because the address of the CFDISC parameter list is zero.</p> <p><b>Action:</b> Verify that the address of the CFDISC parameter list is not corrupted.</p>
0008	0804	<p><b>Equate Symbol:</b> ICFRRCBADVERSIONNUM</p> <p><b>Meaning:</b> A program error occurred because the CFDISC parameter list contains a version number that is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Verify that your program did not overlay the CFDISC parameter list storage.</li> <li>2. Verify that your program was assembled with the correct macro library for the TPF system.</li> </ol>
0008	080A	<p><b>Equate Symbol:</b> ICFRRCBADCONTOKEN</p> <p><b>Meaning:</b> A program error occurred because the requesting processor specified a connect token that is not valid.</p> <p><b>Action:</b> Correct your program to use the original connect token that was received in the CFCONC answer area after the connection request was issued.</p> <p>See “CFCONC–Connect to a Coupling Facility List or Cache Structure” on page 128 for more information about the CFCONC macro.</p>
0008	080B	<p><b>Equate Symbol:</b> ICFRRCCONNINUSE</p> <p><b>Meaning:</b> A program error occurred because the requesting processor tried to disconnect from a CF structure that was still in use.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Let all outstanding requests for the CF structure end.</li> <li>2. Issue the CFDISC macro again.</li> </ol>
0008	08F2	<p><b>Equate Symbol:</b> ICFRRCRESTARTINCOMPLETE</p> <p><b>Meaning:</b> An error occurred because CF restart has not yet completed. The disconnect request is rejected because the TPF system has not yet completed CF restart.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Wait until CF restart has completed successfully.</li> <li>2. Issue the disconnect request again.</li> </ol>

Table 2. Return and Reason Codes for the CFDISC Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
000C	0CF0	<p><b>Equate Symbol:</b> ICFRRCAUTHLOCKERROR</p> <p><b>Meaning:</b> The disconnect request was rejected because the CF lock was not obtained because of an error that occurred. This error occurs when the CF fails to respond to CF commands.</p> <p><b>Action:</b> None.</p>

## Programming Considerations

- You can disconnect from only one CF list or cache structure at a time. If you want to disconnect from multiple CF list or cache structures, issue the CFDISC macro once for each CF list or cache structure.
- Applications should use the deleteCache function to discontinue use of a CF cache structure by a processor rather than using the CFDISC macro. See *TPF C/C++ Language Support User's Guide* for more information about the deleteCache function.

## Examples

None.

## CFISVC—Find Entry in the Macro Information Tables

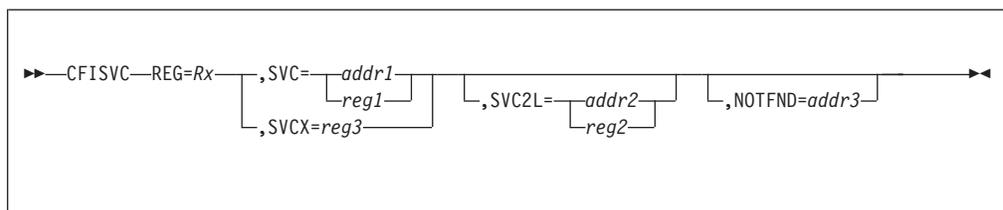
Use this system macro to store the address of an entry in the macro decoder table. The entries in this table are used by the macro decoder itself, test tools, and various dump routines.

Each entry is one of the following:

- Primary supervisor call (SVC)
- Vectored SVC
- Indexed SVC
- A fast link macro.

Each macro must be defined with an appropriate call to the CRESVC macro in the IBMSVC or USRSVC macros. The ISV0SV data macro maps the data at the address provided.

### Format



#### REG=R<sub>x</sub>

This parameter specifies a general register that, upon return from this macro call, contains the location of the SVC table entry for the requested SVC. R14 is not valid.

#### SVC

This parameter specifies the location of the halfword SVC or fast link number. Specify one of the following..

*addr1*

An address.

*reg1*

A register that contains the SVC number.

**Note:** Either the SVC parameter or the SVCX parameter must be coded, but not both.

#### SVCX=*reg3*

This parameter is a register containing the address of an SVC expansion. If this parameter is coded, then CFISVC uses the SVC and index number from the expansion, and returns the address of the corresponding macro information table entry. If the expansion is not an SVC expansion, the error path (see the NOTFND parameter) will be taken. If this parameter is coded, SVC and SVC2L cannot be coded. SVCX and REG cannot use the same register. R14 cannot be used in online code, and R2 is not valid for offline code.

#### SVC2L

This optional parameter contains the location of the halfword index number for the indexed SVC. Specify one of the following:

*addr2*  
An address.

*reg2*  
A register that contains the index number.

**NOTFND=***addr3*

This optional parameter specifies the location where control is transferred if the SVC is not found. If NOTFND is omitted, control is passed to the next sequential instruction (NSI).

## Entry Requirements

None.

## Return Conditions

- REG will contain zero if the SVC number is not found.
- R14 will not be saved when called from C-type or E-type programs. If called from the system test post processor (STPP) or from the in-core dump formatter (ICDF), R2 is not saved across the macro call.

## Programming Considerations

- This macro may be processed on any I-stream online.
- CFISVC is used offline by STPP and online by real-time trace (RTT) among other programs.

## Examples

```
CFISVC SVC=EBW040,REG=R5,NOTFND=ERROR
```

This invocation returns the address in R5 of the macro decoder entry for the SVC specified in ECB location EBW040. If the SVC specified is not found, processing resumes at location ERROR.

```
CFISVC SVC=EBW040,SVC2L=EBW000,REG=R6
```

This invocation returns the address in R6 of the macro decoder entry for the SVC specified in ECB location EBW040 and indexed by the halfword residing in ECB location EBW000.

```
CFISVC SVC=(R3),SVC2L=(R5),REG=R6
```

This invocation returns the address in R6 for the macro decoder entry specified by the SVC in R3 and indexed by the halfword in register R5.

```
CFISVC SVCX=R3,REG=R4,NOTFND=ERROR
```

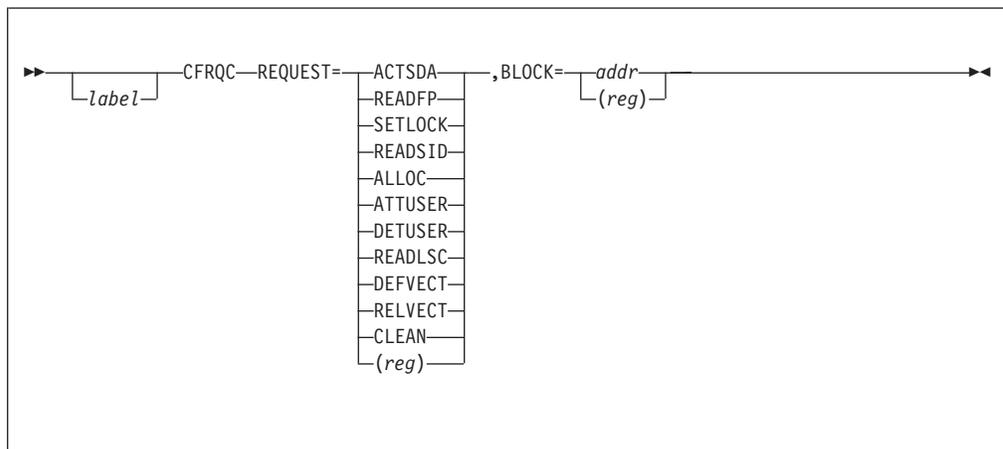
This invocation returns the address in R4 for the macro decoder entry indicated by the SVC expansion addressed by R3. If there is an error, processing resumes at location ERROR.

## CFRQC–Coupling Facility Request

Use this system macro to access coupling facility (CF) control program (CP) routines for entry control block segments that are controlled by the entry control block (ECB). The CF CP routines allow these segments to:

- Activate a CF symbolic device address (SDA)
- Read CF parameters
- Set the CF lock
- Read a structure identifier (ID) vector
- Allocate a CF list structure.
- Attach a list structure user to a CF list structure
- Detach a list structure user from a CF list structure
- Read CF list structure controls
- Define a list notification vector
- Release a list notification vector
- Remove CF structures.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### REQUEST

Indicates the type of request you want to perform. The type must be one of the following values:

#### ACTSDA

Activates the CF SDA.

#### READFP

Reads the parameters for a CF.

#### SETLOCK

Locks or unlocks the CF lock. The *CF lock* is used to serialize operations on the CF.

#### READSID

Allows the list structure user to read the structure ID vector.

#### ALLOC

Allocates a CF list structure.

**ATTUSER**

Attaches a list structure user to a CF list structure.

**DETUSER**

Detaches a list structure user from a CF list structure.

**READLSC**

Reads list structure controls for a CF list structure and returns user structure controls and the user ID vector.

**DEFVECT**

Defines a list notification vector.

**RELVECT**

Releases a list notification vector.

**CLEAN**

Removes CF list structures that are not allocated fully.

**Note:** If you specify ALLOC, ATTUSER, CLEAN, or DETUSER for the REQUEST parameter, the CF lock must be held.

(*reg*)

Register (R0–R7) that contains the value for the type of request you want to perform. The request type must be one of the these values:

ICFREQACTSDA	ICFREQREADFP
ICFREQSETLOCK	ICFREQALLOC
ICFREQATTUSER	ICFREQDETUSER
ICFREQREADLSC	ICFREQDEFVECT
ICFREQRELVECT	ICFREQCLEAN

Equates for these values are defined in the ICFEQ DSECT.

**BLOCK**

Specifies the address of the data block for your request. The format of the input data block is provided with the IFAPI data macro. See that data macro for the correct parameter format.

Specify one of the following:

*addr*

The address of the RS-type name or address of a data block.

*reg*

A pointer to a data block.

**Entry Requirements**

None.

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- The condition code (CC) is not preserved across this macro call.
- The contents of R14 are unknown. The contents of R0–R7 are preserved across this macro call.
- A return code is returned in R15 as follows.

**Request Type Condition Code**

## CFRQC

ACTSDA	ICF_SUCCESS ICF_ERROR
READFP	ICF_SUCCESS ICF_ERROR
SETLOCK	ICF_SUCCESS ICF_ERROR ICF_INVALID_CPUID
ALLOC	ICF_SUCCESS ICF_SUCCESS_AND_ALLOCATED ICF_SUCCESS_AND_RENAMED ICF_NO_FREE_SIDS ICF_SYNC_ERROR ICF_INVALID_STRUCTURE_SIZE ICF_ERROR
ATTUSER	ICF_SUCCESS ICF_ATTACH_FAILED ICF_NAME_NOT_UNIQUE ICF_VECTOR_UNAVAILABLE
DETUSER	ICF_SUCCESS ICF_SUCCESS_AND_DEALLOC ICF_SUCCESS_AND_ZERO ICF_ERROR
READLSC	ICF_SUCCESS ICF_ERROR
DEFVECT	ICF_SUCCESS ICF_FEWER_VECTOR_ENTRIES ICF_VECTOR_UNAVAILABLE
RELVECT	ICF_SUCCESS ICF_TOKEN_NOT_ASSIGNED
CLEAN	ICF_SUCCESS ICF_ERROR

## Programming Considerations

None.

## Examples

- The following shows an example of how to activate a CF SDA:  
CFRQC REQUEST=ACTSDA,BLOCK=ACTSDADB

The data block is found at the location indicated by the ACTSDADB label.

- The following shows an example of how to read the parameters for a CF:  
CFRQC REQUEST=READFP,BLOCK=(R7)

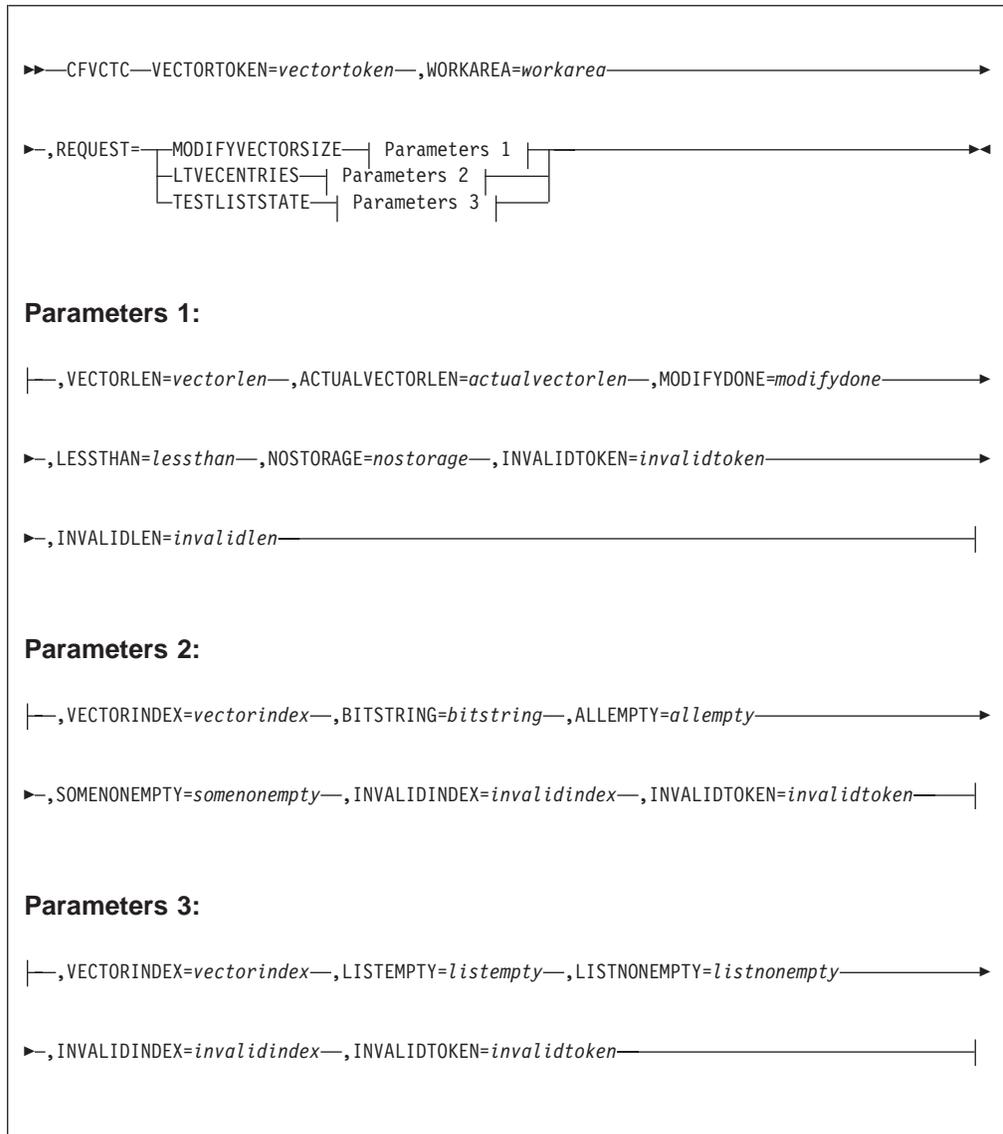
A pointer to the data block is found in register 7 (R7).

## CFVCTC—Check or Modify a List Notification Vector

Use this system macro to perform the following operations on your list notification vector:

- Modify the number of entries in your list notification vector.
- Test a range of list notification vector entries to determine whether the associated lists are empty or not empty.
- Test whether a list is empty or not empty.

### Format



#### VECTORTOKEN=*vectortoken*

Identifies the list notification vector on which you want the specified request (using the REQUEST parameter) performed. This list notification vector was initially created by the CFCONC macro, if the VECTORLEN parameter was coded, and the vector token was returned in the ICFCAAVECTOROK field of the CFCONC answer area. See “CFCONC—Connect to a Coupling Facility List or Cache Structure” on page 128 for more information about the CFCONC macro.

To code, specify the name or address of an 8-byte field that contains the token for the list notification vector.

**WORKAREA**=*workarea*

Specifies a work area that is used by the CFVCTC macro. The minimum required length for the work area is defined by the ICFVWALEN equate. This equate is defined by the IFAPI data macro.

To code, specify the name or address of a work area that begins on a word boundary.

**REQUEST**

Allows you to manage the list notification vector, vector entries, and a list.

Specify one of the following:

**MODIFYVECTORSIZE**

Modifies the size of your list notification vector.

**LTVECENTRIES**

Loads and tests a range of vector entries associated with a list notification vector for a CF list structure.

**TESTLISTSTATE**

Tests whether a list you are monitoring is empty or not empty.

**VECTORLEN**=*vectorlen*

Specifies a fullword field that contains the new total number of vector entries in the list notification vector. The new number of vector entries can be greater than or less than the current number of vector entries, but must be greater than 0. If you specify a number that is not a multiple of 32, the TPF system will round up the number to a multiple of 32.

The TPF system tries to expand or contract your list notification vector to the size you specify.

To code, specify the name or address of the fullword field that contains the requested list notification vector length.

**ACTUALVECTORLEN**=*actualveclen*

Specifies a fullword field to receive the count of the new number of vector entries for the list notification vector after the REQUEST=MODIFYVECTORSIZE parameter is processed. The fullword field is valid only when control is passed to the routine specified by the LESSTHAN parameter.

To code, specify the name or address of the fullword field to contain the new list notification vector length if it differs from the requested list notification vector length.

**MODIFYDONE**=*modifydone*

Specifies the label to branch to if the CFVCTC macro is able to modify the list notification vector.

To code, specify the name or address of the label.

**LESSTHAN**=*lessthan*

Specifies the label to branch to if the CFVCTC macro cannot obtain enough storage to enlarge your list notification vector to the size you requested.

To code, specify the name or address of the label.

**NOSTORAGE**=*nostorage*

Specifies the label to branch to if the CFVCTC macro cannot obtain any storage

## CFVCTC

to enlarge your list notification vector to the size you requested. The list notification vector size remains the same.

To code, specify the name or address of the label.

### **INVALIDTOKEN**=*invalidtokn*

Specifies the label to branch to if the vector token you specified on the VECTORTOKEN parameter is not valid.

To code, specify the name or address of the label.

### **INVALIDLEN**=*invalidlen*

Specifies the label to branch to if the value you specified on the VECTORLEN parameter is not valid.

To code, specify the name or address of the label.

### **VECTORINDEX**=*vectorindex*

If REQUEST=LTVECENTRIES is coded, specifies the starting index of the range of vector entries that you want loaded and tested. The index you specify must be evenly divisible by 32. Thirty-two consecutive vector entries will be loaded and tested. The vector indexes for a specific vector of size N range from 0 to N-1.

If REQUEST=TESTLISTSTATE is coded, this parameter specifies a fullword field that contains the vector index entry associated with the list of interest. For a vector with N entries, valid vector index values range from 0 to N-1.

To code, specify the name or address of the label.

### **BITSTRING**=*bitstring*

This is a required output field that contains vector entry state information for the range of vector entries specified. This field contains 32 bits. The first bit represents the first vector entry specified on the VECTORINDEX parameter and continues up to a maximum of 32 vector index entries. The bits are interpreted as follows:

- 0** The vector entry that corresponds to this bit position indicates that the monitored list is not empty
- 1** The vector entry that corresponds to this bit position indicates that the monitored list is empty.

To code, specify the name or address of a fullword field.

### **ALLEMPTY**=*allempty*

Specifies the label to branch to if all associated lists in the range of the vector entries are empty.

To code, specify the name or address of the label.

### **SOMENONEMPTY**=*somenonempty*

Specifies the label to branch to if some of the associated lists in the range of vector entries are not empty.

To code, specify the name or address of the label.

### **INVALIDINDEX**=*invalidindex*

Specifies the label to branch to if the CFVCTC macro detects that the index value you specified is not valid.

To code, specify the name or address of the label.

**LISTEMPTY=*lempty***

Specifies the label to branch to if the CFVCTC macro finds the list of interest empty.

To code, specify the name or address of the label.

**LISTNONEMPTY=*lstonempty***

Specifies the label to branch to if the CFVCTC macro finds the list of interest not empty.

To code, specify the name or address of the label.

## Entry Requirements

None.

## Return Conditions

When the CFVCTC macro returns control to your program, register 15 (R15) is undefined and all remaining registers are unchanged.

The following tables contain the reasons for which each branch routine may be called and the suggested recovery actions.

*Table 3. Branch Routines for the CFVCTC Macro with REQUEST=MODIFYVECTORSIZE Coded*

Routine	Meaning and Action
MODIFYDONE	<p><b>Meaning:</b> The list notification vector was modified as requested.</p> <p><b>Action:</b> None.</p>
LESSTHAN	<p><b>Meaning:</b> This is a system error. The list notification vector is smaller than the size you requested because there was not enough storage available. The new number of vector entries is returned in the field specified by the ACTUALVECTORLEN parameter.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. If you need a larger list notification vector, try the request later when more storage is available.</li> <li>2. If the problem continues, see your system programmer to determine the cause of the problem and to correct it.</li> </ol>
NOSTORAGE	<p><b>Meaning:</b> This is a system error. Storage could not be obtained to increase the list notification vector size. The size remains unchanged.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. If you need a larger list notification vector, try the request later when more storage is available.</li> <li>2. If the problem continues, see your system programmer to determine the cause of the problem and to correct it.</li> </ol>

Table 3. Branch Routines for the CFVCTC Macro with REQUEST=MODIFYVECTORSIZE Coded (continued)

Routine	Meaning and Action
INVALIDTOKEN	<p><b>Meaning:</b> This is a program error because the vector token you specified on the VECTORTOKEN parameter is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Check your program for errors such as the following: <ul style="list-style-type: none"> <li>• You specified the address of the vector token incorrectly.</li> <li>• The vector token was overlaid or corrupted between the time you received it and the time you specified it.</li> <li>• You disconnected from the CF list structure using the CFDISC macro before issuing the CFVCTC macro. You must be connected to the CF list structure to issue the CFVCTC macro. See “CFDISC–Disconnect from a Coupling Facility List or Cache Structure” on page 143 for more information about the CFDISC macro.</li> </ul> </li> <li>2. Correct the error.</li> <li>3. Run the program again.</li> </ol>
INVALIDLEN	<p><b>Meaning:</b> This is a program error because the list notification vector length you specified on the VECTORLEN parameter is not valid. The list notification vector length must be greater than or equal to 1.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Correct the error by ensuring the list notification vector length specified is greater than 0.</li> <li>2. Run your program again.</li> </ol>

Table 4. Branch Routines for the CFVCTC Macro with REQUEST=LTVECENRIES Coded

Routine	Meaning and Action
ALLEMPTY	<p><b>Meaning:</b> All associated lists in the range of vector entries are empty.</p> <p><b>Action:</b> None.</p>
SOMENONEMPTY	<p><b>Meaning:</b> A list in the range of vector entries is not empty.</p> <p><b>Action:</b> None.</p>
INVALIDINDEX	<p><b>Meaning:</b> This is a program error because the vector index you specified on the VECTORINDEX parameter is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Check your program for errors such as the following: <ul style="list-style-type: none"> <li>• You specified a vector index value that is not a multiple of 32.</li> <li>• You specified a vector index value greater than or equal to the number of vector entries. Vector index values for a vector with N entries range from zero to N–1.</li> <li>• You specified the address of the vector index value incorrectly.</li> </ul> </li> <li>2. Correct the error.</li> <li>3. Run the program again.</li> </ol>

Table 4. Branch Routines for the CFVCTC Macro with REQUEST=LTVECENTRIES Coded (continued)

Routine	Meaning and Action
INVALIDTOKEN	<p><b>Meaning:</b> This is a program error because the vector token you specified on the VECTORTOKEN parameter is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Check your program for errors such as the following: <ul style="list-style-type: none"> <li>• You specified the address of the vector token incorrectly.</li> <li>• The vector token was overlaid or corrupted between the time you received it and the time you specified it.</li> <li>• You disconnected from the CF list structure using the CFDISC macro before issuing the CFVCTC macro. You must be connected to the CF list structure to issue the CFVCTC macro. See “CFDISC–Disconnect from a Coupling Facility List or Cache Structure” on page 143 for more information about the CFDISC macro.</li> </ul> </li> <li>2. Correct the error.</li> <li>3. Run the program again.</li> </ol>

Table 5. Branch Routines for the CFVCTC Macro with REQUEST=TESTLISTSTATE Coded

Routine	Meaning and Action
LISTEMPTY	<p><b>Meaning:</b> This list is empty.</p> <p><b>Action:</b> None.</p>
LISTNONEMPTY	<p><b>Meaning:</b> The list is not empty.</p> <p><b>Action:</b> None.</p>
INVALIDINDEX	<p><b>Meaning:</b> This is a program error because the vector index you specified on the VECTORINDEX parameter is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"> <li>1. Check your program for errors such as the following: <ul style="list-style-type: none"> <li>• You specified a vector index value greater than or equal to the number of vector entries. Vector index values for a vector with N entries range from zero to N-1.</li> <li>• You specified the address of the vector index value incorrectly.</li> <li>• Another unit of work modified the vector size while the REQUEST=TESTLISTSTATE parameter was being processed.</li> </ul> </li> <li>2. Correct the error.</li> <li>3. Run the program again.</li> </ol>

## CFVCTC

Table 5. Branch Routines for the CFVCTC Macro with REQUEST=TESTLISTSTATE Coded (continued)

Routine	Meaning and Action
INVALIDTOKEN	<p><b>Meaning:</b> This is a program error because the vector token you specified on the VECTORTOKEN parameter is not valid.</p> <p><b>Action:</b> Do the following:</p> <ol style="list-style-type: none"><li>1. Check your program for errors such as the following:<ul style="list-style-type: none"><li>• You specified the address of the vector token incorrectly.</li><li>• The vector token was overlaid or corrupted between the time you received it and the time you specified it.</li><li>• You disconnected from the CF list structure using the CFDISC macro before issuing the CFVCTC macro. You must be connected to the CF list structure to issue the CFVCTC macro. See “CFDISC–Disconnect from a Coupling Facility List or Cache Structure” on page 143 for more information about the CFDISC macro.</li></ul></li><li>2. Correct the error.</li><li>3. Run the program again.</li></ol>

## Programming Considerations

- When the CFVCTC macro is called from an entry control block (ECB) routine, all addresses must be ECB virtual memory (EVM) addresses. When this macro is called from a control program (CP), all addresses must be system virtual memory (SVM) addresses.
- The vector indexes are local to the central processing complex (CPC) you are running and do not reside on the CF.

## Examples

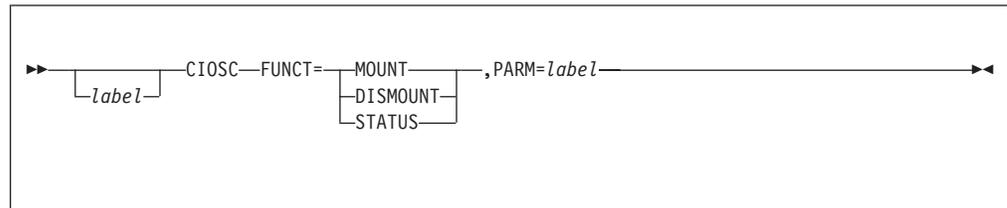
None.

## CIOSC—Request a Mount, Dismount or Status of an SDA

Use this system macro to:

- Mount or dismount a symbolic device address (SDA)
- Request the status of an SDA mount.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **FUNCT**

This is the name of the function to be performed. It must be one of the following values:

##### **MOUNT**

Specified SDA is to be mounted

##### **DISMOUNT**

Specified SDA is to be dismounted

##### **STATUS**

Return the mount status of the specified SDA

#### **PARM**

This is the required parameter for the function to be performed.

##### **MOUNT**

This is a label assigned to an area of storage containing a MDR or a register containing the address of a MDR as defined by the DCTMDR DSECT. See “MSDAC—Mount a Symbolic Device Address (SDA)” on page 356 for more information.

##### **DISMOUNT**

This is the label of a halfword field containing the SDA or a register with the SDA in bytes 2-3 and zeros in bytes 0-1. See the “DSDAC—Dismount a Symbolic Device Address (SDA)” on page 217 for more information.

##### **STATUS**

This is a label assigned to an area of storage containing a DDB or a register containing the address of a DDB as defined by the DCTDDB DSECT. See “ISDAC—Interrogate Symbolic Device Address (SDA) Status” on page 326 for more information.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction.

## CIOSC

- The contents of R14 are unknown. The contents of R0 – R7 are preserved across this macro call.
- A code is returned in R15 as follows.  
MOUNT:
  - 0 SDA mount was successful.
  - 4 SDA is in-use.
  - 8 SDA is not valid.DISMOUNT:
  - 0 Dismount was successful.
  - 4 Dismount was unsuccessful.STATUS:
  - 0 SDA is in-use.
  - 4 SDA is available.
  - 8 SDA is not valid.
- If the extended option is set in the DDB (DDBX = 1) for STATUS, the path management control words and the subchannel status words are returned in the area defined by the DCTDDB DSECT.
- The path available mask is returned on a successful mount in the area defined by the DCTMDR DSECT.
- For MOUNT and STATUS requests, the channel path identifier (CHPID) type indicator is returned.

## Programming Considerations

- This macro can be run only on an I-stream having affinity with the specified SDA.
- An SDA specification that is not valid for a dismount results in a return code of 0.
- This macro is for use in a real-time program only.

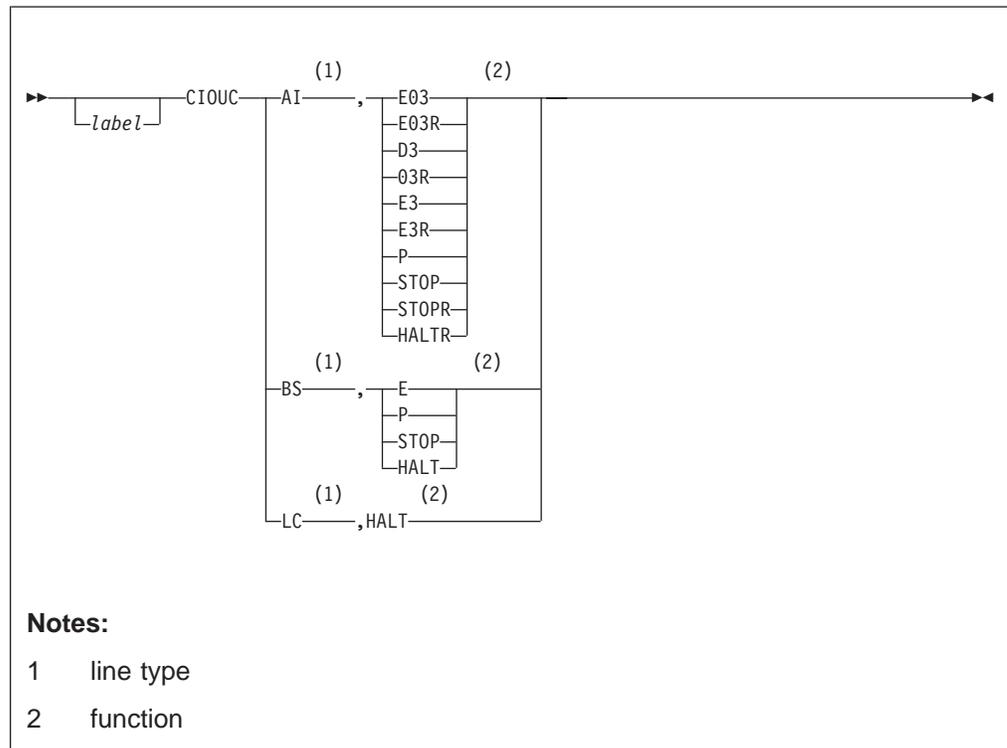
## Examples

None.

## CIOUC—Initialize and Reset Communication Lines

Use this system macro to initialize and reset or stop input/output (I/O) on any communication line.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*line type*

Type of communication line must be specified as parameter one.

**AI** ATA/IATA SLC lines (SLC)

**BS** Binary Synchronous Lines (BSC)

**LC** 3270 Local Lines (LC)

*function*

Initialize and reset functions required must be specified as parameter two.

**E** Enable

**E03** Enable 3705 EP send side

**E03R** Enable 3705 EP receive side

**D3** Disable 3705 EP send side

**03R**

Disable 3705 EP receive side

**E3**

Disable then enable 3705 EP send side

**E3R**

Disable then enable 3705 EP receive side

**P** Prepare or Poll

**R** Release

**HALT**

Stop I/O to end a Prepare or Poll command.

**HALTR**

Stop I/O to end a Prepare or Poll command on the receive side.

**STOP**

Stop I/O due to a suspected error condition. On 3705 EP stop on send side.

**STOPR**

Stop 3705 EP I/O due to a suspected error condition on receive side.

**Notes:**

1. Some of the previous functions are invalid for certain line types.
2. Depending on the line type, a combination of functions could be performed by using more than one symbol for parameter two.
3. On some of the functions, it is possible to concatenate an R suffix, to denote that the operation applies to a Receive subchannel.
4. The following is a list of the function combinations and valid line types.

- ATA/IATA SLC (IBM 3705 EP):

AI, E03	Enable only, send side
AI, E03R	Enable only, receive side
AI, D3	Disable only, send side
AI, 03R	Disable only, receive side
AI, E3	Disable then enable, send side
AI, E3R	Disable then enable, receive side
AI, P	Issue Prepare command
AI, STOP	Halt I/O due suspected error, send side
AI, STOPR	Halt I/O due suspected error, receive side
AI, HALTR	Halt I/O to end Prepare command

- BSC:

BS, E	Enable
BS, P	Issue Prepare or Poll command
BS, STOP	Halt I/O due to suspect error
BS, HALT	Halt I/O to end Prepare or Poll command

- 3270 Local:

LC, HALT	Halt I/O to end activity
----------	--------------------------

## Entry Requirements

R14 must contain the symbolic line number (SLN).

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of scratch registers R14 and R15 are unknown. The contents of the remaining operational registers and the condition code are saved during processing of this macro.

## Programming Considerations

- This macro is used exclusively by the communications control program.
- The CIOUC macro should only be used on the main I-stream.
- The ECB reference register (R9) contains the address of the ECB before this macro is used.
- The symbolic line number given in R14 is checked. If the line number does not fall within the range of lines in the TPF system a system error is generated and the ECB is forced to EXIT.
- The status of the CIOUC operation can never be determined by the initiating program.
- This macro is for use in the control program (CP) only.

## Examples

None.

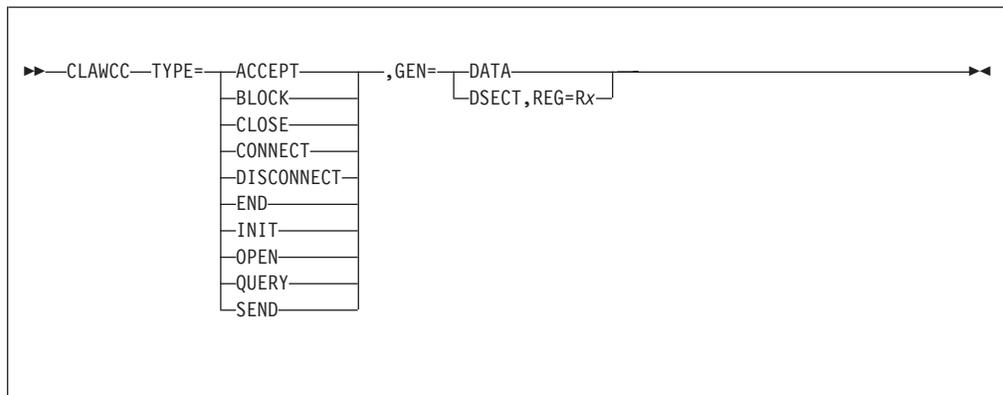
## CLAWCC–CLAW API Linkage

Use this system macro to perform the following services:

- Define the code generation for interfacing with Common Link Access to Workstation (CLAW) application programming interface processing in the control program (CP).
- Generate a dummy control section (DSECT) to describe the code generation.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### TYPE

The CLAW service to be performed.

#### ACCEPT

Accepts a connection request

#### BLOCK

Starts a CLAW block function for GCLAC or RCLAC

#### CLOSE

Shuts down a logical link

#### CONNECT

Requests a CLAW connection to a remote workstation

#### DISCONNECT

Disconnects a connection to a remote workstation

#### END

Stops CLAW activity

#### INIT

Prepare for CLAW activity

#### OPEN

Initialize an adapter

#### QUERY

Get the status of an adapter or logical links

#### SEND

Send a message on an active logical link

**GEN**

Indicates the type of generation required.

**DATA**

Generates the code to link to the CLAW service routines.

**DSECT**

Generates a DSECT that defines the linkage code associated with the generation type.

**REG=Rx**

Specifies a register for DSECT generation. This parameter is required if you code GEN=DSECT.

**Entry Requirements**

None.

**Return Conditions**

The generated code returns to the calling routine using the linkage information in R2. The contents of R2 are unknown.

**Programming Considerations**

- This macro can be run on any I-stream. However, CLAWCC is a restricted use macro and is intended only for use by the CLAW library functions.
- This macro cannot be implemented inline. To avoid reentry problems, macro generation is moved to a stack area modified by the routine and branched to by R2.
- If you code GEN=DSECT, you must also specify a register for the DSECT by using REG=Rx.

**Examples**

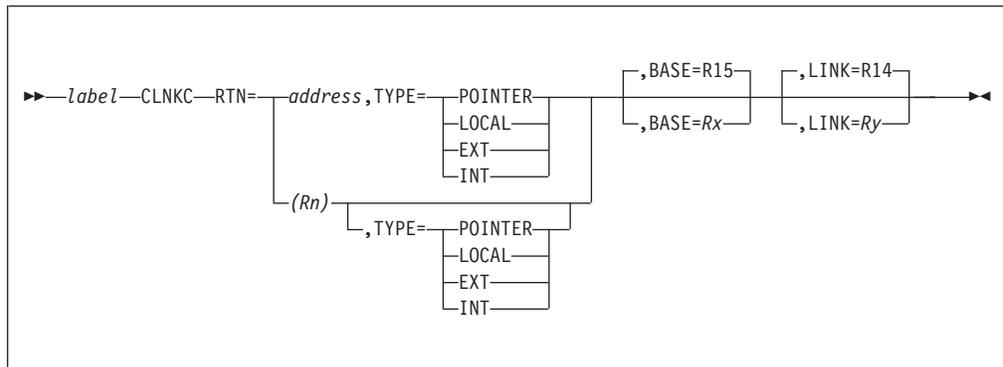
- This call generates the code necessary to link to the CLAW ACCEPT service routine.  
CLAWCC TYPE=ACCEPT,GEN=DATA
- This call generates the DSECT used for the CLAW ACCEPT service routine using register R1.  
CLAWCC TYPE=ACCEPT,GEN=DSECT,REG=R1

## CLNKC–Control Program (CP) Call and Link

Use this system macro to generate the standardized linkage to call a control program (CP) routine. This macro is used with other standardized linkage macros such as the DLNKC, RLNKC, and SLNKC macros. See the following for more information about these macros:

- “DLNKC–Define Stack DSECT for Control Program (CP) Routine” on page 216
- “RLNKC–Return to CP Calling Routine and Reset Stack Pointer” on page 409
- “SLNKC–Control Program (CP) Save Link Data & Set Stack Pointer” on page 448.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BASE=R15|R<sub>x</sub>**

The register specified, default R15, will be used as the link-to register between the routines. The called routine will use this register as its base register.

#### **LINK=R14|R<sub>y</sub>**

The register specified, default R14, will be used as the link-from register between the routines. The called routine will use this register as the return register. This register is NOT saved and restored over the call.

#### **RTN=address|(R<sub>n</sub>)**

This parameter can be either a label or a register specified as (R<sub>n</sub>). This parameter must be the base of the called routine except when TYPE=POINTER. In this case, the parameter will point to the address of the routine. If the register specification is used and the TYPE parameter is omitted, the proper address is assumed to be in the register. If the address specification is used, the TYPE parameter must be specified as EXT, INT, LOCAL, or POINTER.

#### **TYPE**

This parameter specifies the type of addressability this routine has to the called routine identified by the RTN parameter. If the TYPE parameter is not specified, the required address is assumed not to be a POINTER. The TYPE parameter must be specified when the RTN parameter specifies an address.

#### **POINTER**

The called control program (CP) routine’s address is a pointer.

#### **LOCAL**

The called control program (CP) routine’s address is within the current base register addressability.

**EXT**

The called control program (CP) routine's address is located in a different CSECT than this routine. The address is resolved through a Vcon.

**INT**

The called control program (CP) routine's address is within the current CSECT but not within the current base register addressability. The address is resolved through an Adcon.

## Entry Requirements

R13 must contain the correct stack pointer.

## Return Conditions

- Control is normally returned to the next sequential instruction, but need not be.
- The contents of the registers are determined by the interface between this routine and the called routine.

## Programming Considerations

- This macro can be run on any I-stream.
- R13 must point to a valid stack.
- The condition code is determined as part of the interface between the two routines.
- The program invoked by this macro should use the other linkage and stack manipulation macros (SLNKC, DLNKC, RLNKC) to save and restore the proper registers and stack base.
- This macro is for use in the control program (CP) only.
- The contents of the register specified by BASE will be saved on the stack and restored by the called routine.

## Examples

None.

---

## CPDSC—Generate Control Program (CP) DSECT

Use this system macro to produce the low core DSECT area for common reference by the control program (CP) CSECTS. The content is dependent on conditional code in the macro.

### Format



*name*  
The symbolic name of the CSECT.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- I-stream restrictions are not applicable to this macro.
- This macro is for use in the control program (CP) only.

### Examples

None.



## CPLKC

### **RMRESUME**

Performs the `xa_start`, `flags=TMRESUME` process. RMRESUME requires the RMID as a parameter.

### **RMPREP**

Performs the `xa_prepare`, `flags=TMNOFLAGS` process. RMPREP requires the RMID as a parameter.

### **RMROLLBK**

Performs the `xa_rollback`, `flags=TMNOFLAGS` process. RMROLLBK requires the RMID as a parameter.

### **RMCOMMIT**

Performs the `xa_commit`, `flags=TMNOFLAGS` process. RMCOMMIT requires the RMID as a parameter.

### **DEBUG\_RTN**

Returns to the TPF Assembler Debugger for VisualAge Client intercept point. No parameter is required for this function.

### **CDEBUGPER**

Requests the program event recording (PER) facility bracket override for the TPF Assembler Debugger for VisualAge Client.

### **ADD\_PET**

Adds the program entry to the PER exclusion table for the TPF Assembler Debugger for VisualAge Client. The ADD\_PET function requires the program allocation table (PAT) slot address as a parameter.

### **REMOVE\_PET**

Removes the program entry from the PER exclusion table for the TPF Assembler Debugger for VisualAge Client. The REMOVE\_PET function requires the PAT slot address as a parameter.

### **SETASMBKPT**

Sets the breakpoint table in the TPF Assembler Debugger for VisualAge Client.

### **ADD\_LOB**

Adds a program entry to the load occurrence breakpoint table. The ADD\_LOB function requires an entry structure pointer as a parameter.

### **REMOVE\_LOB**

Removes a program entry from the load occurrence breakpoint table. The REMOVE\_LOB function requires an entry structure pointer as a parameter.

### **INITASMDBG**

Initializes the TPF Assembler Debugger for VisualAge Client.

### **MQPREMT**

Performs the internal `pre_commit` function (which performs preliminary functions necessary for ending the current transaction scope) for an MQSeries resource manager.

### **TMADDCCR**

Adds a commit control record (CCR) to a specified resource manager that is participating in a transaction. The TMADDCCR function requires a pointer to the CCR and RMID as parameters.

### **PARMLIST**

An optional parameter list is defined by each function. The CPLKC service routine will not verify the parameter list.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R0–R7 are preserved across this macro call. The contents of all other registers are unknown.
- The condition code (CC) is not preserved across the macro call.
- The macro service routine will set R14 to contain a hexadecimal return code that can be used by the calling program. Return conditions may vary depending on the function requested. See the *TPF C/C++ Language Support User's Guide* for more information about the `xa_start`, `xa_end`, `xa_prepare`, `xa_commit`, `xa_rollback`, and `xa_recover` C functions and the related return codes.
- A return code is not set for the `DEBUG_RTN`, `ADD_PET`, and `REMOVE_PET` functions.
- A return code of 0 for the `ADD_LOB` function means that the entry was added successfully to the load occurrence breakpoint table. A return code of 1 for the `ADD_LOB` function means that the entry was not added to the load occurrence breakpoint table because the entry is already in the table. A return code of 2 means there is not enough heap storage to add another entry.
- A return code of 1 for the `REMOVE_LOB` function means that the entry was not found in the load occurrence breakpoint table. The entry structure contains the entry that was removed.

## Programming Considerations

This macro can be run from any I-stream.

## Examples

The following shows an example of a process using commit processing.

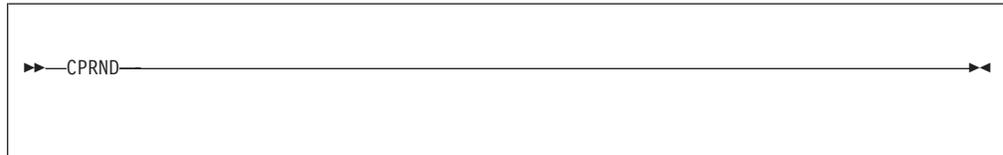
```
.
CPLKC FUNCTION=TMSTART
.
```

---

## CPRND–Control Program (CP) Rounding

Use this system macro to increase the size of a control program (CP) CSECT to the next 256-byte boundary.

### Format



### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- This macro should be placed in a control program (CP) SECT segment after all code statements.
- This macro rounds up the size of a CSECT to the next 256-byte boundary. This is done to keep the size relatively constant as code changes and for patching purposes.
- It identifies the source of the object code produced by placing the assembly date, the assembly time and the CSECT name in the object code just ahead of the 256-byte boundary.

**Note:** In many installations it is customary to define the CSECT name with a version number.

- This macro is for use in the control program (CP) only.

### Examples

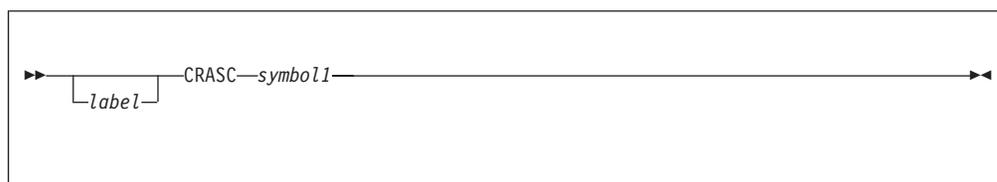
None.

## CRASC—Send Message to CRAS

Use this system macro to send a message to the computer room agent set (CRAS).

**Note:** The CRASC macro should not be used by system programs and applications. System programs and applications must use the ROUTC macro. All replies must be made with the WTOPC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*symbol1*

A data level in the ECB must be specified as parameter one.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- Messages for the computer room console must be contained in a 381-byte block.
- The message format for the computer room console follows here.

	Standard Header__	Character Count*__	Symbolic Address=	Text	E O M
Bytes:	0----15	16---17	18----20	21	-

Where:

- \* The character count includes the 3-byte device address, the text field, and the EOM character.
- = This field contains the symbolic address of the computer room console. For the RO (Receive Only) CRAS bytes 18 through 20 should contain the value X'000000'. For the prime CRAS bytes 18 through 20 should contain the value X'010000'. The appropriate physical device address will be resolved by the control program (CP).
- The messages must be in internal EBCDIC, including function characters such as New Line (#LFEED), which serves the purpose of carriage return and line feed. The EOM character serves no functional purpose for a 1052 terminal and is not a required special character. However, for a clear indication of end-of-message and for message compatibility, the normal high-speed EOM character (#EOM) should be used.
- Bit 1 of CE1CPA must be set to 1 for the send intercept routine to be bypassed for the messages that already have the 3270 data stream format.
- Only valid 1977 characters should be used in the message as 1052 CRAS terminal is to fallback to 1977 terminal. See *TPF Programming Standards* for more information about the character sets provided. The reason for this restriction is that when a 1052 prime CRAS, 1052, 3215, or 3270 Local CRT is

## CRASC

inoperable the TPF system looks for a substitute. First it tries an alternate 1052/3215 from the line status table. If not available, it tries for a local 3270 CRT and RO. If not available as well, it looks for a 1977/2915/4505 or 3270 with RO printer in the CRAS Status Table. Because a 1977/2915/4505 terminal might be used during extraordinary error conditions the messages should be limited to characters valid for these devices.

- The messages destined for the Send Intercept Routine must ensure that the CE1FMx field associated with the input message is zeroed.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The specified data level (CBRW) is available.

## Programming Considerations

- This macro can be run from the main I-stream only.
- The message to be sent is contained in a storage block held by the entry control block.
- A check is made by the control program (CP) to determine whether the ECB is holding a block of storage at the specified level. If a block is not held, control is transferred to the system error routine.
- The block of storage containing the message to be sent is no longer available to the operational program.
- The status of the sending operation can never be determined by the operational program.
- The operational program may use the specified level immediately upon return from this macro.

## Examples

Send message to the prime CRAS.

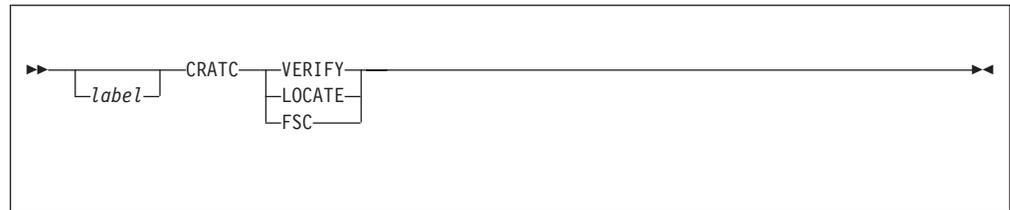
```
      GETCC      D3,L1
      MVC        16(CRASCMSGL-CRASCMSG-1,R14),CRASCMSG
      CRASC      D3
      .
      .
CRASCMSG DC AL2(CRASMSGL-CRASMSG-3)
          DC XL3'010000'
          DC C'TEXT OF THE MESSAGE',AL1(#EOM)
CRASMSGL EQU *
```

## CRATC–Search CRAS Status Table

Use this system macro to:

- Verify that a terminal address (line number, interchange address, and terminal address (LNIATA)/CPUID) is in the CRAS status table (CROAT)
- Obtain the LNIATA and CRAS table slot address of:
  - A functional support console (FSC)
  - An alternate CRAS workstation given an alternate CRAS slot number or associated printer index.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### VERIFY

Is used to determine if a given LNIATA/CPUID is present in the CRAS table.

#### LOCATE

Is used to find the CRAS slot address for one of the following:

- A functional support console when given a routing code indicator and a CPU ID or
- A an alternate CRAS terminal when given an Axx slot number or a printer index number.

#### FSC

Is used to find the CRAS slot address of a functional support console when given the routing index located at the beginning of the CRAS table.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- R0 must contain the input in one of the following formats:
  - For the VERIFY option:
 

Bytes	0-2	contains LNIAT
	3	contains CPUID (optional)
  - For the LOCATE option:
    - To return the CRAT slot address for each functional support console (FSC) type as follows.
 

Bytes	0	contains zeros
	1-2	contains console type (as defined in RTCEQ)
	3	contains CPUID (optional)
    - To return the CRAT slot address for each printer index or alternate CRAS slot number (Axx) as follows.
 

Bytes	0	contains the index
	1-3	contains zeros
  - For the FSC option:

## CRATC

Bytes 0 contains the index  
1-3 contains zeros

If the CPUID is omitted, byte 3 of R0 must contain zeros and the CPUID will default to that of the ECB (CE1CPD).

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The following output is placed in general register 0:
  - For VERIFY option:
    - If the LNIATA is not found, R0 will contain zeros.
    - If the LNIATA is found, bytes 0-2 of R0 will remain unchanged. Byte 3 of R0 will contain the CPUID used in processing the macro request (unchanged if a CPUID was supplied on input; CE1CPD if the CPUID was omitted).
  - For the LOCATE and FSC Options:
    - If the FSC/printer/Alternate CRAS is found, R0 will contain the following.  
Bytes 0-2 contains LNIATA  
3 contains CPUID (optional)
    - If a request is made for an FSC that is not valid or an unused printer/Axx slot, R0 will contain zeros.
- R15 will contain the CRAS table slot address.
- The contents of R14 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- The CRATC macro expands to a fast link macro for E-type segments; it expands to a BASSM for C-type segments.
- This macro requires 6 bytes of storage when issued by a E-type segment and 10 bytes of storage when issued by a C-type segment.
- The CRATC macro should not be issued from a C-type segment, which is interruptible.
- General register 9 (R9) must contain the address of the ECB before processing the macro.
- General register 0 (R0) is used for all input and output for this macro.
- If a successful search is completed, general register 15 (R15) will contain the storage address of the CRAS table slot.

## Examples

Reference macro RTCEQ for the functional support console definitions.

Bit	Console Type	Label
0	RO	RTCDRO
1	PRC	RTCDPRC
2	TAPE	RTCDTA
3	DASD	RTCDDA
4	COMM	RTCDCOM
5	AUDIT	RTCDAUD

Bit	Console Type	Label
6	TPFAR	RTCDRDB
7–15		Reserved for Use by IBM

Usage examples:

- Verify that the terminal with LNIATA 4A1713 is in the CRAS table as follows.

Input: R0 = 

4A	17	13	00
----	----	----	----

Code: CRATC VERIFY

Output: R0 = 

4A	17	13	C2
----	----	----	----

#### Return Code Meaning

R0 Shows that 4A1713 is in the CRAS table with an ECB CPUID of B.

R15 The CRAT slot address of terminal 4A1713B.

- Determine where the COMM console is located in CPU A as follows.

Input: R0 = 

00	08	00	C1
----	----	----	----

Code: CRATC LOCATE

Output: R0 = 

51	00	00	C1
----	----	----	----

The LNIATA of the COMM console is returned in the high-order 3 bytes of R0.

#### Return Code Meaning

R15 The CRAT slot address of terminal 510000A.

- To return a LNIATA given a printer/Axx slot number (for example, A10) as follows.

Input: R0 = 

0A	00	00	00
----	----	----	----

Code: CRATC LOCATE

Output: R0 = 

LN	IA	TA	X
----	----	----	---

 where X = CPUID

#### Return Code Meaning

R15 The CRAT slot address of terminal LNIATAX.

- To find a FSC using the routing code indices located at the beginning of the CRAS table (label CR0ACRI in CR0AT) as follows.

Input: R0 = 

16	00	00	00
----	----	----	----

Code: CRATC LOCATE

Output: R0 = 

LN	IA	TA	X
----	----	----	---

 where X = CPUID

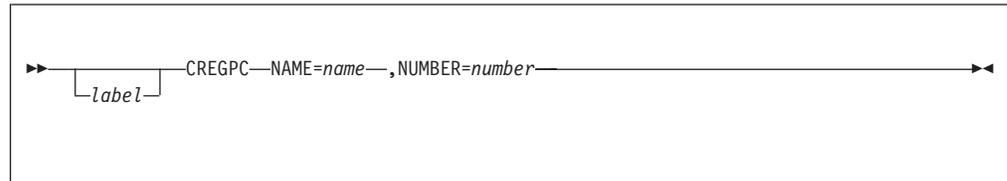
## CRATC

<b>Return Code</b>	<b>Meaning</b>
R0	Shows that LNIATAX is the terminal address of the FSC in the CRAS table slot A22 (X'16').
R15	The CRAT slot address of terminal LNIATAX.

## CREGPC—Create a Macro Group Definition

Use this system macro to create a macro group definition. Macro groups permit you to set debugger macro breakpoints to trap a group of supervisor calls (SVC) or fast link macros.

### Format



*label*

is a symbolic name that can be assigned to the macro statement.

**NAME=*name***

specifies the macro group, where *name* is the 8-character macro group name.

**NUMBER=*number***

specifies the number that is assigned to the macro group, where *number* is a macro group number (0 through 16 for IBM groups or 17 through 32 for user groups) that is associated with the GRPCODE or ADDGRP parameter on the CRESVC macro.

### Entry Requirements

This macro can be coded only in the IBMSVC or USRSVC macros.

### Return Conditions

None.

### Programming Considerations

None.

### Examples

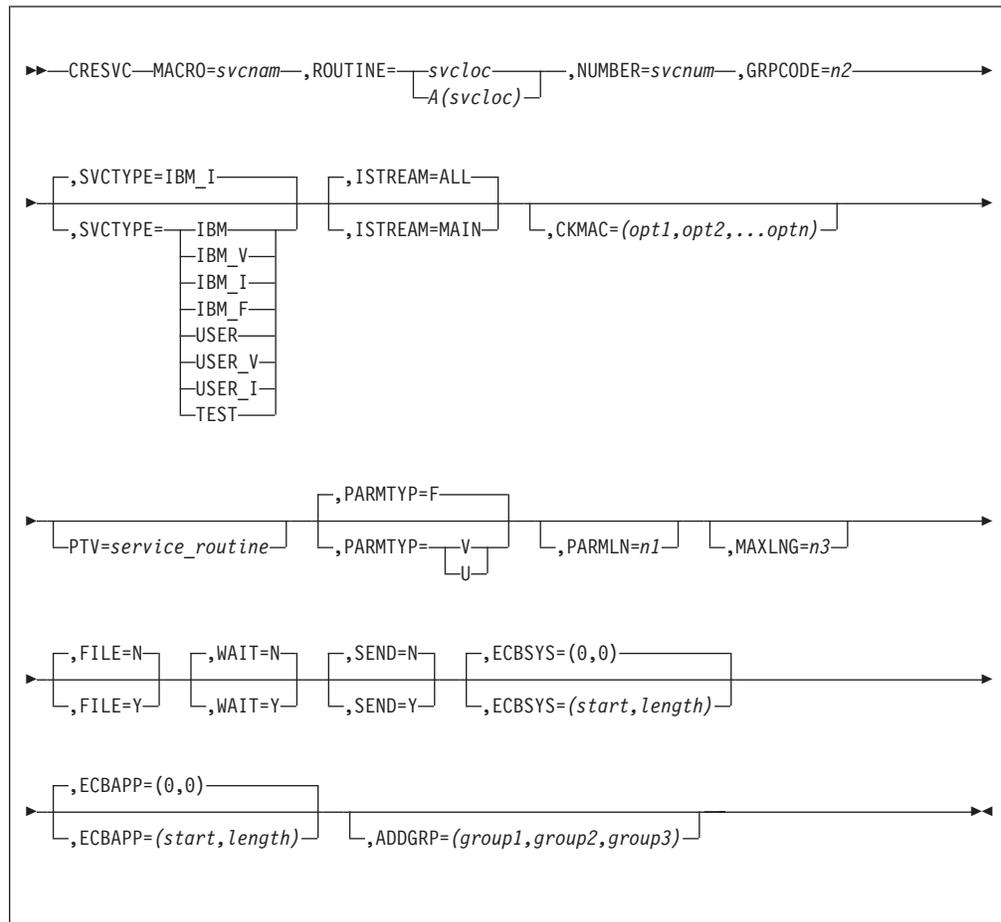
The following example creates a macro group definition called find; the find macro group definition is associated with a test tool group code that is specified with the GRPCODE parameter on the CRESVC macro.

```
CREGPC NAME=FIND,NUMBER=6
```

## CRESVC—Create an SVC/Fast-Link Table Entry

Use this system macro to centralize the process of adding a macro interface system service to the TPF system. This macro generates data necessary for defining to the TPF system a primary or indexed supervisor call (SVC), or a fast link macro.

### Format



#### MACRO=*svcnam*

This parameter specifies the macro name for the SVC. Specify a one- to six-character macro name for SVCs and a five- or six-character name for fast-link macros. If the names are not unique, an assembly error will be generated. For fast-link macros, the first four letters of the name must be different.

#### ROUTINE

This parameter specifies the location of the SVC service code. For SVCs that are pointers to indexed tables (when SVCTYPE=IBM\_V or USER\_V), this parameter is optional since the address of the driver is given automatically. This parameter is also optional for SVCTYPE=TEST as well.

#### *svcloc*

A VCON to the entry point for the macro service is generated.

#### A(*svcloc*)

An address constant is generated for the SVC location.

**NUMBER=***svcnm*

This parameter specifies the SVC, index or fast-link number. It is the index into the appropriate macro table as defined by the SVCTYPE parameter. The macro definition is automatically placed in the proper macro table in the appropriate entry:

- For indexed tables, the range of numbers for a table with *n* sequential elements must be 0 to *n*-1 with no gaps; otherwise, an error is generated.
- For indexed SVCs, the highest value of *svcnm* is 32767.
- For user SVCs, the numbers are restricted to numbers 97, 127, 162, and odd numbers from 197 to 253.
- For primary and vectored SVCs, the number must be in the range 1-255. A vectored SVC is one that points to the driver code for an indexed table.
- For fast-link macros, the number specified must be between 256 and 32767.
- The numbers from 500 to 599 are reserved for user specification (that is, SVCTYPE=USER\_F).

**CKMAC=(***opt1,opt2,...optn***)**

This parameter is used to specify that this macro requires an authorization check when it is called by an E-type program. Valid options are:

**RESTR**

Restricted use macro check

**KEY0**

Key0 write authorization check

**MONTC**

Supervisor state authorization check

**CMB**

Common block authorization check.

One or more of these options may be specified in any order. If no options are specified, the parameter is ignored. See “\$CKMAC–Validate Use of Restricted Macro” on page 14 for more information.

**PTV=***service\_routine*

This parameter specifies the PTV service routine. The address is switched with the address specified by the ROUTINE parameter when PTV is active. There is no default.

**PARMTYP**

This parameter specifies the type of parameter(s) to be passed.

- F** The parameter length is fixed. This is the default.
- V** The parameter length is variable. The length is contained in a halfword following the SVC number (for primary SVCs) or the SVC index number (for indexed SVCs).
- U** The parameter length is unknown. It is to be calculated by the service routine. Use PARMTYP=U for fast-link type macros.

**PARMLN=***n1*

This parameter specifies the parameter length used with fixed parameter types (PARMTYP=F). The length does not include the SVC opcode nor the SVC number. Specify a decimal number in the range 0-32767.

**SVCTYPE**

This parameter specifies the table in which the macro definition will reside.

## CRE SVC

The following are supported macro service types for IBM-defined macros and their supporting service routines.

### **IBM**

An IBM-defined SVC for the primary table (0-255).

### **IBM\_V**

An SVC that points to an indexed table containing IBM-defined SVC services. Vectored SVCs must have PARMTYP=U.

### **IBM\_I**

An IBM-indexed SVC. This is the default.

### **IBM\_F**

A fast-link SVC (numbers 256-32767 except 500-599).

The following are supported macro service types for user-defined macros and their supporting service routines.

### **USER**

A user-defined SVC for use in the primary table (restricted to numbers 97, 127, 162, and odd numbers from 197 to 253).

### **USER\_V**

An SVC that points to an indexed table that contains user SVCs (restricted to numbers 97, 127, 162, and odd numbers from 197 to 253).

### **USER\_I**

A user-indexed SVC.

### **USER\_F**

A user-defined fast-link macro. Numbers 500-599 are reserved for users.

Used by the Test Tools functions.

### **TEST**

SVCs that are used by test tools to overlay the fast-link macros to enable proper tracing and/or trapping. These can only be used in the primary table.

### **ISTREAM**

This parameter specifies whether the SVC runs on the main I-stream or all I-streams.

### **ALL**

Indicates that the SVC runs on all I-streams. This is the default.

### **MAIN**

Indicates that the SVC runs on the main I-stream only.

### **GRPCODE=*n2***

This parameter specifies the proper test tools group code. See the *TPF Program Development Support Reference* for details. This is an optional parameter for TEST, IBM\_V and USER\_V SVC types, but is mandatory for all other types. The user groups can be assigned as additional groups in IBMSVC by using the ADDGRP parameter. Specify a decimal number from 0 to 15. There is no default.

### **MAXLNG=*n3***

This optional parameter specifies the maximum parameter length, in bytes, to be dumped in RTT. Specify a number from 0 to 8.

**FILE**

This optional parameter indicates if the macro is a file macro or unit record macro. RTT uses this parameter to determine whether a data block should be dumped when the macro is traced.

**N** Do not dump the data block. This is the default.

**Y** Dump the data block.

**WAIT**

This optional parameter indicates if the SVC macro implies a wait (for example, FINWC and FIWHC). It is used exclusively by real-time trace (RTT).

**N** This macro does not imply a wait. This is the default.

**Y** This macro implies a wait.

**SEND**

This optional parameter indicates if the macro is a communication send type for which real-time trace (RTT) should dump data blocks.

**N** This macro is not a send type, data blocks should not be dumped. This is the default.

**Y** This macro is a send type, data blocks should be dumped.

**ECBSYS=(start,length)**

This optional parameter specifies the ECB data area associated with a given macro to be dumped by Realtime Trace (RTT). The starting location in the ECB and the length (in bytes) of the area to be dumped can be specified by an ECB field name or can be a hardcoded, decimal number. The default is ECBSYS=(0,0). For example,

(CE1FA0,CE1SUG+1-CE1FA0)

This specifies that RTT should begin dumping at ECB location CE1FA0 and continue dumping between CE1FA0 and CE1SUG (inclusive).

**ECBAPP=(start,length)**

This optional parameter specifies the ECB data area associated with an application to be dumped by Realtime Trace (RTT). The starting location in the ECB and the length (in bytes) of the area to be dumped can be specified by an ECB field name or can be a hardcoded, decimal number. The default is ECBAPP=(0,0). For example,

(EBW000,EBW064-EBW000)

This specifies that RTT should begin dumping at ECB location EBW000 and continue dumping between EBW000 and EBW064.

**ADDGRP=(group1,group2,group3)**

This optional parameter allows the macro to be assigned to more than one group. The additional groups are used by the macro breakpoint support for TPF C Debugger for VisualAge Client and TPF Assembler Debugger for VisualAge Client. A maximum of three additional groups can be specified, up to a total of four groups. The group numbers must be from 0 to 32. The additional groups are not used by RTT.

## Entry Requirements

- There will be one CRESVC for each primary SVC, each indexed SVC and each fast link macro defined for the TPF system.

## CRESVC

- This macro must only be used within the IBMSVC and USRSVC macros. These macros define the application programming interface (API) between the users and system services. Serious errors can occur if this macro is given semantically incorrect (but syntactically correct) parameters.

See “IBMSVC—Generate IBM SVC and Fast-Link Tables” on page 282 and “USRSVC—Generate the User SVC Tables” on page 504 for more information about the IBMSVC and USRSVC macros, respectively.

## Return Conditions

None.

## Programming Considerations

- Thirty-two primary SVC numbers are reserved for users. See the NUMBER parameter for more information about SVC number restrictions.
- This is a declarative macro.
- Only one vector SVC is permitted for each valid vector SVCTYPE (that is, one for SVCTYPE=IBM\_V and one for SVCTYPE=USER\_V).
- This should not be used outside the proper location in the CCMCDC CSECT.
- CRESVC places the specified data in the primary, indexed, or fast-link table at the location determined by the NUMBER parameter.
- If TYPE=EQU is coded for IBMSVC or USRSVC, but CRESVC creates only the defined equate for the macro.
- For indexed SVC macros, the equate contains both the primary number and the index number.

## Examples

- Example of an IBM-defined SVC:

```
CRESVC MACRO=FINWC,ROUTINE=CPMFIW,NUMBER=38,
      SVCTYPE=IBM,PARMTYP=F,PARMLN=2,
      GRPCODE=05,MAXLNG=02,WAIT=Y,ECBSYS=(CE1FA0,CE1SUG+1-CE1FA0)

CRESVC MACRO=PROGC,ROUTINE=CCECPROG,NUMBER=000,
      SVCTYPE=IBM_I,PARMTYP=F,PARMLN=10,
      GRPCODE=11,MAXLNG=10,CKMAC=RESTR
```
- Example of a fast-link SVC:

```
CRESVC MACRO=TIMEC,NUMBER=383,GRPCODE=11,MAXLNG=02,
      ROUTINE=CPTMRT,SVCTYPE=IBM_F
```
- Example of a user SVC:

```
CRESVC MACRO=HVSVC,ROUTINE=LOCATION,PARMTYP=V,NUMBER=162,
      SVCTYPE=USER,GRPCODE=3,MAXLNG=5,ECBSYS=(CE1FA0,CE1SUG1+1-CE1FA0)
```
- Example of a user indexed SVC:

```
CRESVC MACRO=SUNCC,ROUTINE=CICRRRR,PARMTYP=V,SVCTYPE=USER_I,
      NUMBER=68,ISTREAM=MAIN,GRPCODE=5,MAXLNG=5,ECBAPP=(EBW000,
      EBW064-EBW000)
```

## CROSC–Cross-Subsystem Access Service Request

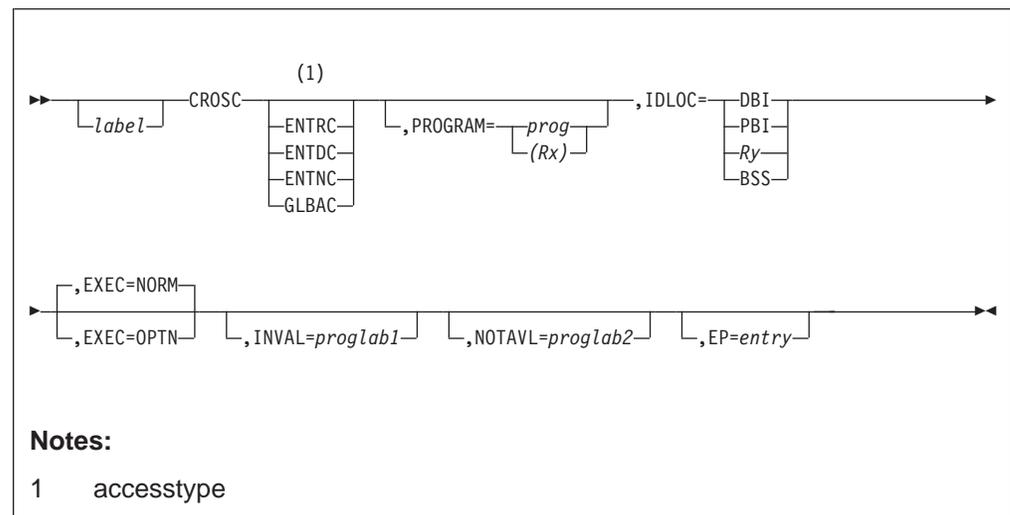
Use this system macro to obtain various types of functional access to a user's program base or database. The access types are:

- Program access
- Global area database access.

In a multiple database function (MDBF) environment, the CROSC macro is the interface between common or user-shared TPF programs and unique user-associated data and programs. Through the macro services facilities, user-sensitive TPF programs can access programs and databases that are necessarily user-dependent.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *accesstype*

A positional parameter specifying the access type to be defined or obtained. This is required in all macro calls.

#### **ENTDC**

Enter a program and release all previous programs attached to the ECB.

#### **ENTNC**

Enter a program with no return expected.

#### **ENTRC**

Enter a program with return expected (not allowed from control program (CP) when EXEC=NORM is specified).

#### **GLBAC**

Obtain subsystem global area update service (not allowed from the control program (CP)).

**PROGRAM**

The program name that is to be accessed when the access type is ENTDC, ENTNC, or ENTRC.

*prog*

The name of the program that is to be entered.

*(Rx)*

A register (R0-R7) that contains the address of where the program name can be found.

**IDLOC**

A keyword operand specifying the location of the identifier to be used in selecting the table access definitions. This is required in all macro calls.

**DBI**

Use database ID field (CE1DBI)

**PBI**

Use program base ID field (CE1PBI)

*Ry* Use the value in the specified register. When *Ry* is coded, it must be the symbolic name of (R0–R7) that contains the 2-byte MDBF identifier if the *accesstype* was GLBAC. Otherwise, it contains the 2-byte MDBF identifier for the target PBI. Note that not all valid MDBF identifiers are valid PBIs.

**BSS**

Use the BSS (Basic Subsystem) program base.

**INVAL=*proglab1***

A keyword operand identifying the routine label to be given control if the identifier fails the MDBF ID integrity check (ordinal number plus complement not equal to X'FF'), if the designated subsystem was not included in the last IPL, or if the designated subsystem is inactive. This operand is *required* if IDLOC=*Rx* and EXEC=OPTN are specified in the macro call. The operand is *optional* in other macro calls, but if it is omitted and an error condition is raised, the ECB will be exited with a system error (for integrity check failure, or for inactive or not included in last IPL).

**NOTAVL=*proglab2***

A keyword operand identifying the routine label to be given control if the CROSC global interface program, GLBL, has not been locked in storage by segment GOGO. If not defined and the condition is raised, a system error with exit is taken.

**EP=*entry***

A keyword operand specifying the desired entry point into the global area update program. This operand is *required* in macro calls requesting global access (access type GLBAC). Entry point equates are defined in the equate macro GLBEQ.

**EXEC**

Specify one of the following:

**OPTN**

OPTN specifies that optional servicing of the specified access request is needed in lieu of normal servicing. This is optional in macro calls requesting program base access. If OPTN is specified, control is returned to the caller with R14 containing the address of the enter expansion; the enter expansion is not processed.

OPTN is not available when doing a global area database access.

**NORM**

If NORM is specified or left to default and the program is a form of Enter, the requested function is activated.

**Entry Requirements**

- The control program (CP) can request program base access (except through ENTRC), but not global area access.
- The control program (CP) can request a program base access without supplying the address of an ECB in R9. To accomplish this IDLOC=Rx and EXEC=OPTN must be specified in the macro call.
- If the operand IDLOC=Rx is coded, the general register specified by Rx must contain the identifier to be used in selecting the access table.
- If you want control returned to an error routine when the identifier contained in Rx is incorrect or valid but not available, you must specify the INVAL parameter.
- If you want control returned to an error routine when the function requested is not available, you must specify the NOTAVL parameter.
- The entry must not have a record held if update to CE1DBI is requested.
- The entry reference register (R9) contains the address of the ECB prior to using this macro if the service is requested by an ECB level entry.

**Return Conditions**

- Normal Return:
  - If EXEC=NORM is coded and the function is ENTRC, ENTNC, or ENTDC, then activate the requested function.
  - Return to next sequential instruction (NSI) after processing the GLBAC functions.
  - If EXEC=OPTN is coded return to the NSI with a condition code of 0 and R14 set to the address of the ENTER expansion in the CROSC expansion.
  - The content of R15 is unknown.
  - The contents of the remaining general registers are saved during processing of this macro.
  - The ECB fields CE1DBI, CE1PBI, CE1SSU\*, CE1GLA\*, and CE1GLY\* are updated according to the following matrix:

FUNCTION TYPE	EXEC=OPTN				EXEC=NORM			
	IDLOC=				IDLOC=			
	Rx	DBI	PBI	BSS	Rx	DBI	PBI	BSS
PROG. ACCESS ENTR/ENTN/ ENTD	–	DBI → PBI	–	–	(Rx) → DBI → PBI	DBI → PBI	–	BSS → PBI
GLOBAL ACCES 'GLBAC'	N/A	N/A	N/A	N/A	(Rx) → DBI	–	PBI → DBI	N/A

**Note:** Whenever the field CE1DBI (DBI) is modified the value in CE1DBI will be used to locate, and decrease by one, the number of active ECBs for

## CROSC

that subsystem. The value with which it is being replaced will be used to locate, and increase by one, the number of active ECBs for the subsystem.

In conjunction with the modification of CE1DBI, the field CE1SSU will be changed to reflect the identification of the first or only subsystem user of the subsystem in question. The values in CE1GLA and CE1GLY will be changed to reflect the base and address of GLOBAL areas A and Y for the subsystem user.

- Error Returns:
  - If the INVAL operand is supplied and the SS ID is found to be incorrect or not included in this IPL, return to the next sequential instruction with CC=1. R14 will be set to zero if the SS ID is incorrect, and nonzero if the SS ID is not included in the IPL. If the error routine address is not specified, a system error is taken.
  - If the NOTAVL operand is supplied and the requested function is not defined for the subsystem, return to the next sequential instruction with CC=2. If the error routine address was not specified, a system error is taken.

**Note:** All system errors are with EXIT if the calling routine is an ECB level entry, otherwise the system error is catastrophic.

## Programming Considerations

- This macro can be run on any I-stream.
- The CROSC macro should be used only when it is necessary to cross subsystem boundaries.
- ENTRC to FACE or FACS is a special type of enter and can not be used on the CROSC macro. Attempting to CROSC to FACE or FACS will result in a system error dump.
- The CROSC access type ENTNC cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES).

## Examples

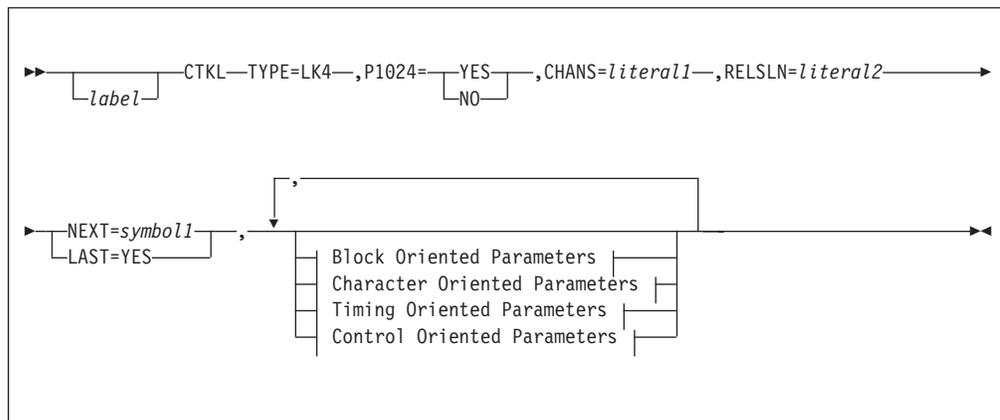
None.

## CTKL-SLC Channel Keypoints Setup

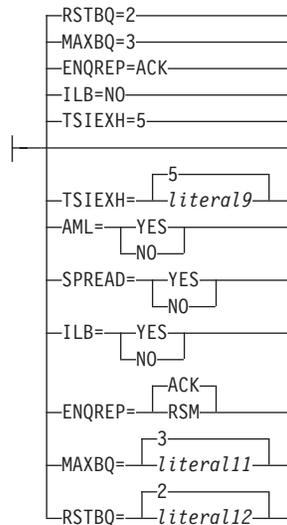
Use this system macro to generate the link keypoint (LK4KC) and channel keypoints (LK5LC) that are required to operate and control TPF synchronous links. These keypoints are stored on file as program records and assembled by using one or more CTKL statements.

### Format

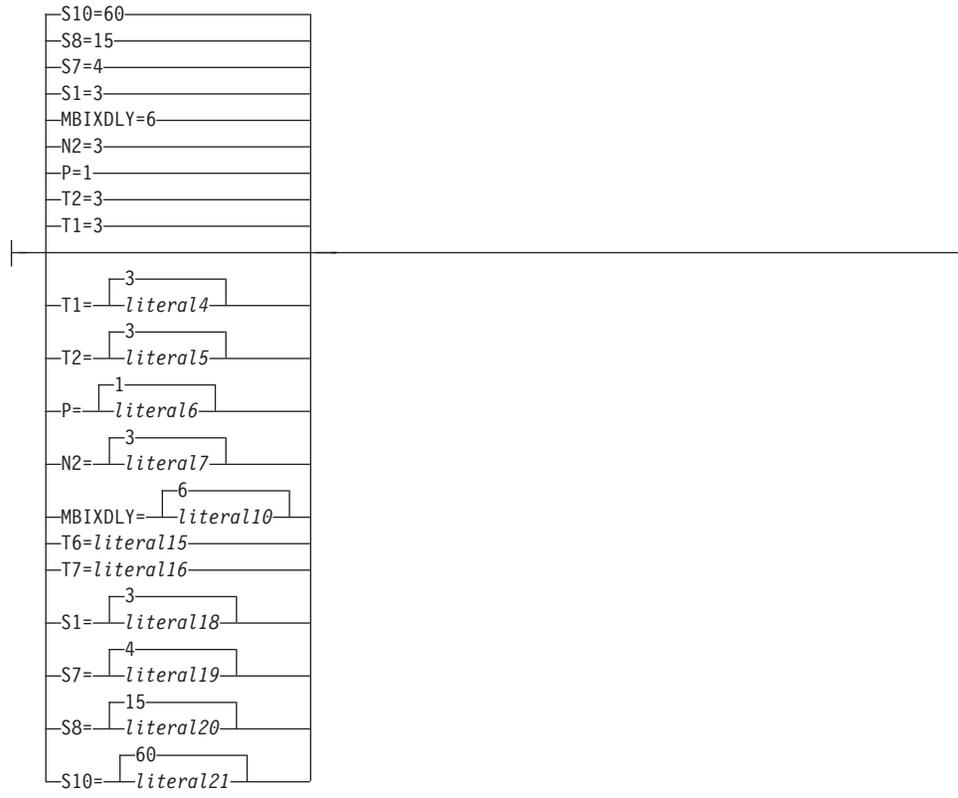
If you want to generate a link keypoint, use the following format. The parameters required in each case are listed separately.



#### Block Oriented Parameters:



**Timing Oriented Parameters:**



**Character Oriented Parameters:**



**Control Oriented Parameters:**

Z5=3
ORIGID=0000
RETRANS=NO
HDR= YES
NO
ORIGID= 0000
literal13
DESTID=literal14
ACI= OMIT
INCLUDE
LINCODE= CCITT#5
CCITT#2
EXITT#5
ALC
Z5= 3
literal22
LOOPST= YES
NO
LOOPBST= ONE
ZERO
RETRANS= NO
YES

Each program record (segment) comprises one link keypoint and hence only one CTKL macro is required within the envelope of BEGIN/FINIS statements.

**TYPE=LK4**

Causes expansion of a link keypoint.

**P1024=YES|NO**

Defines whether the link operates according to the SITA link control procedure P1024 or the ATA/IATA synchronous link control contained in ATA/IATA interline communications manual DOC/GEN 1840, Chapter V.

**CHANS=literal1**

Defines the number of full duplex pairs (AI lines) comprising the link. Its value can range from 1 to 7.

**RELSLN=literal2**

All lines attached to the TPF system are allocated sequential symbolic line numbers defined in SYCON. Literal 2 defines the lowest numbered AI line assigned to this link relative to the minimum symbolic line number assigned to AI lines in SYCON. The lowest RELSLN value is 0 (zero).

**NEXT=symbol1****LAST=YES**

These parameters are mutually exclusive and either provide, through symbol1, the name of the next program record (segment) in the chain of keypoints or else indicate, through LAST=YES, that this program record is the last one in the chain.

**MAXBLK=literal3**

Defines the maximum number of characters in data message blocks exchanged on the link from DLE to ETB inclusive. It does not include BCC. The default value is 255.

**T1=literal4**

Defines the NAK/ENQ/Data Procedure timeout and repetition frequency. The default value is 3 seconds.

**T2=literal5**

Defines the idle line timeout and repetition frequency. The default value is 3 seconds.

**P=literal6**

Defines the frequency of acknowledging data message blocks (notwithstanding the fact that all blocks of multiblock messages will be acknowledged individually). The default value is 1.

**N2=literal7**

Defines the number of consecutive idle line or ENQ timeouts that cause the line to be out of service (LOS). The default value is 3.

**HEN=literal8**

If the link is connected at its other end to a high level network (for example SITA HLN) literal8 must be four hexadecimal characters defining the address of entry centre (HEN) to that network. This parameter must be included if P1024 = YES, otherwise a default value of zero is assumed.

**TSIEXH=literal9**

Defines the number of outstanding TSIs not acknowledged that causes the TPF system to inhibit data message block transmission and enter the enquiry procedure (TSI Exhaustion). The default value is 5.

**AML=YES|NO**

Defines whether AML link control blocks are exchanged when all blocks of a multiblock message have been correctly received and acknowledged to clear that message label for reuse by the transmitting center.

If the parameter is omitted, the value assigned depends on the parameter P1024:

- P1024=YES generates AML=YES
- P1024=NO generates AML=NO.

If P1024=YES and AML=NO is coded, the parameter is accepted and a warning message generated.

**SPREAD=YES|NO**

Defines whether blocks of multiblock messages can be transmitted on different channels of a multichannel link (also called *scatter*).

If the parameter is omitted, the value assigned depends on the parameter P1024:

- P1024=YES generates SPREAD=NO
- P1024=NO generates SPREAD=YES.

If P1024=YES and SPREAD=YES is coded, the parameter is accepted and a warning message generated.

**ILB=YES|NO**

Defines whether ILB link control blocks are exchanged in idle line conditions. The default value is ILB=NO

If P1024=YES and ILB=YES is coded the parameter is accepted and a warning message generated.

**ENQREP=ACK|RSM**

Defines whether the response to ENQ and ILB link control blocks are ACK in addition to RSM or STP or are restricted to RSM or STP. The default value is ENQREP = ACK.

**EXSYN=\*|4|8|12**

Defines the number of extra SYN characters at the start of a data transmission additional to the number generated automatically by the hardware. The default value is 0 (zero).

**MBIXDLY=*literal*10**

Defines the time delay after discovering MBI exhaustion for A-type messages before attempting further processing by the output message handler. The default value is 6.

**MAXBQ=*literal*11**

Defines the maximum number of message blocks on the link B-type output message queue which inhibits the output message handler from adding further original message blocks. The default value is 3.

**RSTBQ=*literal*12**

Defines the minimum number of message blocks on the link B-type output message queue which allows the CCP to control transfer to the output message handler. The default value is 2.

**HDR=YES|NO**

Defines whether all A-type traffic on this link, with the exception of traffic that originated from a pseudo high-speed line, contains routing information in a standard message header, or not.

If this parameter is omitted, it is assumed that the message will not contain routing information.

**ORIGID=*literal*13**

This parameter defines the symbolic origin application name assigned to nonPLN, A-type traffic on a link that does not support the message routing header. Literal 13 must be four decimal characters. A default of 0000 is generated if this parameter is required, but not included.

**DESTID=*literal*14**

This parameter defines the symbolic destination application name assigned to nonPLN, A-type traffic on a link that does not support the TPF message routing header. Literal 14 must be four decimal characters. A default of 0000 is generated if this parameter is required, but not included.

**ACI=OMIT|INCLUDE**

Defines whether the optional additional characteristics indicator (ACI) byte is to be included in the control character envelope which accompanies each information block. If the parameter is omitted, the value assigned is dependent upon the P1024 specification:

- P1024=NO generates ACI=OMIT
- P1024=YES generates ACI=INCLUDE.

When actually coding the ACI parameter, the validity of the definition is dependent upon the P1024 parameter. Therefore, P1024=NO, ACI=OMIT, P1024=NO, ACI=INCLUDE, and P1024=YES, ACI=INCLUDE are valid combinations while P1024=YES, ACI=OMIT are not valid combinations.

**LINCODE=CCITT#5|CCITT#2|EXITT#5|ALC**

Defines the code translation to be performed on message block text when the

optional ACI character is not a component of the control character envelope (ACI=OMIT). If the parameter is omitted, the value generated is LINCOTE=CCITT#5. The text translation specifications follow here.

CCITT#5	CCITT No. 5 code (ASCII/ISO 7 bit code)
CCITT#2	Padded CCITT No. 2 code (Padded Baudot)
EXITT#5	Extended CCITT No. 5 code
ALC	Padded 6 bit code (Padded ALC).

Only one translation code can be specified for each link when using this parameter.

**T6=literal15**

Defines the time interval allowed between receipt of the last ACK of a multiblock message and the acknowledge message label (AML) LCB before the entire message will be retransmitted. T6 is an alternate value to T1 and overrides the latter when defined. The T6 parameter is invalid for P1024 = YES.

When the alternate AML timer, T6, is defined, the value must be between 1 and 63 inclusive. When T6 is not to be defined, omit the parameter.

**T7=literal16**

Defines the time interval allowed between receipt of successive blocks of a multiblock message before the partially received message is discarded. This parameter is invalid for P1024 = YES.

When a message discard timer value, T7, is used, the value must be between 1 and 63 inclusive. When T7 is not used, omit the parameter. This implies that partially received messages will not undergo a "time out" and be discarded.

**N3=literal17**

Defines the number of times a multiblock message is retransmitted in its entirety when no Acknowledge Message Label (AML) LCB has been received. This parameter is invalid for P1024 = YES.

When message retransmission is to occur, N3 must be a value specified between 1 and 12 inclusive. When no message retransmission is to occur, the N3 parameter must be omitted.

**S1=literal18**

No ACK received timer. After a message block has been sent, an ENQ is generated by the link control handler if an ACK is not received within S1 seconds. The S1 timer is reset after an LCB with parity errors is received and after a message block has been sent. Default value is 3.

**S7=literal19**

No block received timer. When no data or control block has been received for more than S7 seconds, the link control handler generates an ENQ. The S7 timer is reset upon receipt of a control or data block and after an ENQ has been generated. Default value is 4.

**S8=literal20**

Channel down timer. When a channel is declared to be down, the link control handler starts the channel down timer (S8). This timer is reset should the link control handler declares the link down. While the S8 timer is running, the link control handler does not generate or accept data or control blocks. When the timer expires an ENQ is generated. Default value is 15.

**S10=literal21**

Multiblock message timer. If, during reception of a multiblock message S10

seconds have elapsed during which another block of that message has not been received, the link control handler gives to the supervisor those blocks so far received and clears the message label. Default value is 60.

**Z5=literal22**

STP N/M repetitions. If, after a channel has been declared down, other channels of the link remain operative, the link control handler sends Z5 SXTs at intervals of T1 seconds. Default value is 3.

**LOOPST=YES|NO**

To describe whether or not a loop test configuration between two centers will be established. Default is no.

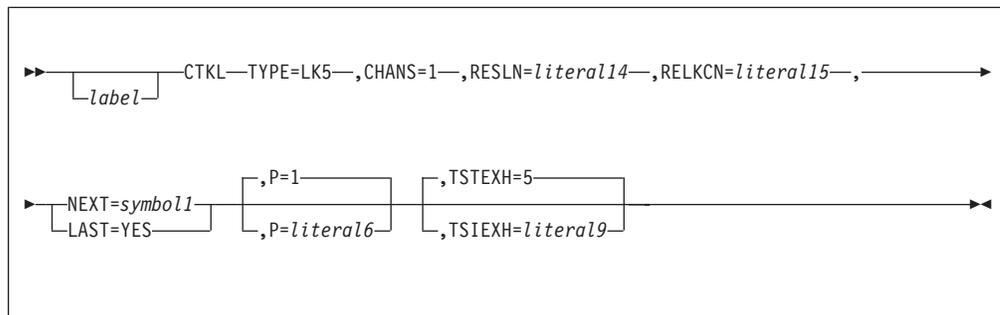
**LOOPBIT=ONE|ZERO**

Subject to bilateral agreement between centers, bit 6 of the TSI in link control blocks can be used to indicate the center originating the control block. It is set to 1 at one end of the link and to 0 (zero) at the other.

**RETRANS=YES|NO**

Indicator as to whether or not to retransmit complete message on channel down condition. Default is no.

If you want to generate a channel keypoint, use the following format.



Each program record (segment) makes up one channel keypoint. Hence, only one CTKL macro is required between BEGIN and FINIS statement.

**TYPE=LK5**

Causes expansion of one channel keypoint

**CHANS=1**

Defines one channel keypoint to be generated by this CTKL statement.

**RELSLN=literal14**

Literal14 defines the lowest numbered AI line whose keypoint is being generated by this CTKL statement relative to the minimum symbolic line number assigned to AI lines in SYCON. The lowest RELSLN value is 0 (zero).

**RELKCN=literal15**

Literal15 defines the lowest numbered AI line whose keypoint is being generated by this CTKL statement relative to the first AI line comprising the link. The lowest RELKCN value (for the first channel assigned to this link) is 0 (zero).

**NEXT=symbol1**

**LAST=YES**

See the previous section for information about the RELKCN parameter and generating a LINK keypoint.

## CTKL

**Note:** When more than one CTKL statement is necessary in one program assembly, each requires this parameter.

**P=literal 6**

See the previous section for information about the P parameter and generating a LINK keypoint.

**Note:** When more than one CTKL statement is necessary in one program assembly, each requires this parameter.

**TSIEXH=literal9**

See the previous section for information about the TSIEXH parameter and generating a LINK keypoint.

**Note:** Where more than one CTKL statement is necessary in one program assembly, each requires this parameter.

## Entry Requirements

SYGLB, SYSET, SYCON, and CAIEQ must be called before issuing the CTKL macro.

## Return Conditions

This is not an executable macro, therefore return conditions are not applicable.

## Programming Considerations

- Only one link keypoint or up to three channel keypoints can be accommodated in one program segment.
- Prior to loading the program segments containing the CTKL statements it is necessary to allocate file storage space for them. This is performed by assigning four character segment names and inserting them into the system allocator.
- If these constitute the first special program records in the system, the programmer must ensure that the name of the first segment in the chain is correctly assembled into the CIJH program. If these records extend to an existing chain, the forward chain field of the previous last-in-chain must be updated to reflect the new next-in-chain.
- The macro defining the Global 3 Directory (GLOBY) must be updated to reflect these special program records.
- I-stream restrictions are not applicable to this macro.

## Examples

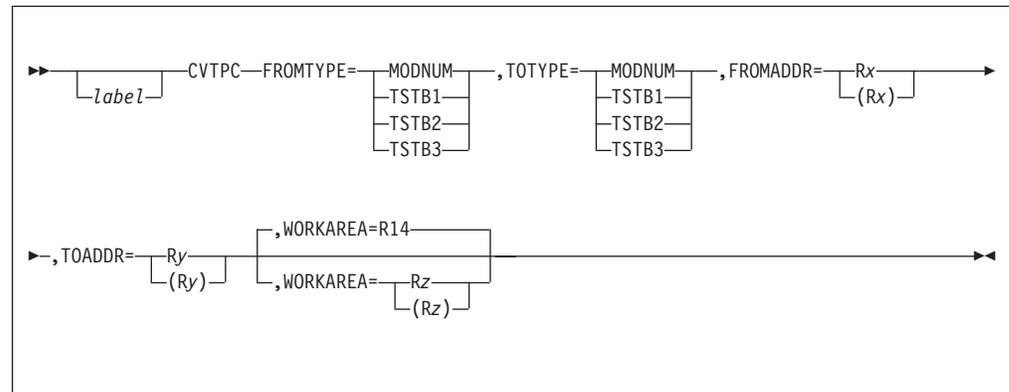
None.

## CVTPC—Convert Tape Status Table Pointer

Use this system macro to convert a tape status table (the ITSTB DSECT) pointer of one type to a pointer of another type. There are 4 types of pointers:

- Module number pointer
- Tape status table section 1 pointer, which provides access to device address, volume serial number (VSN), symbolic tape name, and tape status
- Tape status table section 2 pointer, which provides access to error counters, channel status words (CSWs), tape densities, device types, module queue pointers, and error recovery information
- Tape status table section 3 pointer, which provides access to the main storage copy of the tape label directory and tape label maintenance records, and buffer control information.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FROMTYPE**

Specifies the type of pointer to be converted.

#### **MODNUM**

Converts from a module number

#### **TSTB1**

Converts from a Tape Status Table section 1 pointer

#### **TSTB2**

Converts from a Tape Status Table section 2 pointer

#### **TSTB3**

Converts from a Tape Status Table section 3 pointer

#### **TOTYPE**

Specifies the type of pointer to which the given pointer is converted.

#### **MODNUM**

Converts to a module number

#### **TSTB1**

Converts to a Tape Status Table section 1 pointer

#### **TSTB2**

Converts to a Tape Status Table section 2 pointer

## CVTPC

### TSTB3

Converts to a Tape Status Table section 3 pointer

### FROMADDR

Specifies the fullword pointer to be converted. Parentheses distinguish whether the specified register contains the pointer or the address of the pointer.

Rx Indicates the pointer is in the specified register

(Rx)

Indicates the register points to the specified pointer

### TOADDR

Specifies the converted fullword pointer. Parentheses distinguish whether the specified register contains the pointer or the address of the pointer.

Ry Indicates the pointer is in the specified register

(Ry)

Indicates the register points to the specified pointer

### WORKAREA

Specifies a two word work area that can be used during the conversion. Parentheses distinguish whether the register specifies an even-odd pair or point to a doubleword location in storage.

Rz Indicates an even-odd register pair

(Rz)

Indicates the register points to a doubleword in storage.

R0 cannot be specified for the WORKAREA parameter.

## Entry Requirements

- R0 cannot be specified for the WORKAREA parameter.
- The same register cannot be specified for the WORKAREA and TOADDR parameters.
- The registers specified for the WORKAREA and TOADDR parameters must satisfy several conditions:
  - If WORKAREA is Rx then TOADDR cannot be (Rx).
  - If WORKAREA is Rx then TOADDR cannot be (Rx+1).
  - If WORKAREA is (Rx) then TOADDR cannot be Rx.
  - If WORKAREA is (Rx) then TOADDR cannot be (Rx).

## Return Conditions

- The contents of the register pair or the doubleword specified by the register of the WORKAREA parameter are unpredictable when this macro ends.
- The contents of all registers (except for the WORKAREA register) are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro does not perform validity checking for input parameters. Ensure that all values specified fall in acceptable ranges for the parameter in question.

## Examples

None.

## CWRTC—Write Critical Message to the System Console

Use this system macro to to:

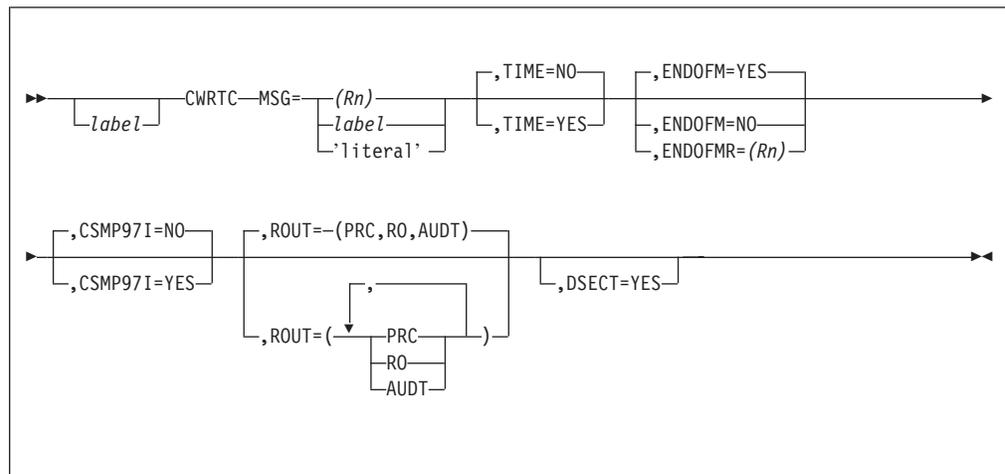
- Send critical messages to the system consoles
- Write critical messages to the system consoles through the common input/output (I/O) priority request (PIOFC).

Mainline communications queuing is bypassed and the messages are written through the preemptive I/O facility. You can call this macro only in C-type (control program) modules. All CWRTC messages are preceded by the CSMP97I header if the SYSTC SBPRMSG switch is set.

Normal I/O for the central processor complex (CPC) is suspended until the console write operations are complete.

The macro function varies slightly, depending on whether the system includes 1052/3215 console support, or (3270) native console support. If 1052 support has been generated, then the macro will write to the 1052 console, and sound the alarm. If native console support has been generated, then the macro will write to the 327x CRT system console, sound the alarm, and also copy the message on the associated 328x printer (RO).

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### MSG

It must be coded in one of the following ways;

MSG=(register)

MSG=label

MSG='literal'

#### (register)

The specified general register must point to the message formatted as below:

DC AL1(Length of message)

DC C' .. message text .. '

The TEXTONLY option of the GENMSG macro may be used to generate the text in the required format.

*label*

If coded, then at that location must be the message formatted as above.

*'literal'*

The macro will generate (inline) a correctly formatted message.

#### **TIME=YES|NO**

It may be coded as YES or NO. NO is the default. If YES is specified, an 8-byte time stamp (of the HH.MM.SS format) will be inserted ten bytes into the message text.

If NO is specified or defaulted, the message will be sent unaltered.

#### **ENDOFM**

It may be coded as YES or NO. If neither ENDOFM nor ENDOFMR is specified, YES is the default. This parameter only affects 1052/3215 console support. This parameter is mutually exclusive with the ENDOFMR parameter.

##### **YES**

An end-of-message-complete character (EOMC) is added at the end of the message.

##### **NO**

An EOMC character is not added at the end of the message.

#### **ENDOFMR=(*Rn*)**

This parameter only affects 1052/3215 console support. If the right-most byte of the specified register is 0, an EOMC character is added at the end of the message. Otherwise, no EOMC is added at the end of the message. This parameter is mutually exclusive with the ENDOFM parameter.

#### **CSMP97I=YES|NO**

Specifies whether the message will be preceded by the CSMP97I header if the appropriate SYSTC switches are set. This facility is intended for use when writing multiline messages. The default is CSMP97I=YES.

CSMP97I=NO should not be coded for the first line of a message, unless the calling program has verified through the SYSTC switches that the CSMP97I header is not required.

#### **ROUT=(code1 [,code2 [,code3 ]])**

Up to three routing codes, as defined in the Functional Support Console Routine Codes segment (RTCEQ), may be specified. This facility is intended for use in writing status messages which need not be logged to all three operator consoles. Only the codes PRC, RO and AUDT are supported. The default is ROUT=(PRC,RO,AUDT).

#### **DSECT=YES**

This form of the macro is used to generate a DSECT which describes the macro parameters. It is intended for use in the system error CSECT only.

## **Entry Requirements**

- General register 14 must be available.
- The TPF system should be masked so that no I/O interrupts will be processed during the processing of this macro.

## CWRTC

### Return Conditions

- Control is returned to the next sequential instruction.
- The condition code is unknown.
- There is no indication as to whether the attempted console write was successful.
- The contents of R14 is unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro must be processed on the main I-stream only.
- This macro may be called only by the main storage resident control program (CP), and should be used only for messages of the highest priority. This macro is for use in the control program (CP) only.
- General register 14 is used in the macro expansion and may not be used to specify the address of the message or as the ENDOFMR register.
- The storage key in the PSW must be such that alteration of protected storage is permitted (that is, Key 0 or F).
- The length of the message may be no more than 255 characters. If an EOMC is to be added (see the ENDOFM/ENDOFMR parameters), the message may be no more than 254 characters. If CSMP97I=YES is coded, the message may be no more than 200 characters.
- If native console support is included, or the 1052/3215 console has been replaced by a local 3270 type device through the ZACRS command, then the message will be reformatted appropriately.
- Only locally attached console devices are supported by the CWRTC macro, that is, subchannel attached 1052/3215, native console, and local 3270s. Network devices are NOT supported. The macro service routine will determine whether or not the console is a network device, and if it is, no message will be sent.
- The CWRTC macro cannot be issued until after the required tables have been initialized by segment CT60 of CTIN. (LINEQ, SLSTL, CR0AT, CX7CW, CX8CW)
- If the MSG parameter is coded as (register), the macro will generate relocatable code. If the MSG parameter is not coded as (register), the macro expansion will include a relocatable address constant.

### Examples

- An example showing a warning message with the identifying message code WARN96W and text WARNING. The time shown by 00.00.00 is replaced by the actual time the message is issued.

```
CWRTC MSG='WARN96W 00.00.00 WARNING',TIME=YES
```

- An example showing a line of message text with no time stamp. An end of message character is appended to the end of the text.

```
CWRTC MSG='JUST LIKE THIS',TIME=NO,ENDOFM=YES
```

- An example showing the use of register R0 to point to the message text.

```
LA    R0,ERROR
CWRTC MSG=(R0)
.
.
.
ERROR DC AL1(L'ERRMSG)           LENGTH OF THE TEXT
ERRMSG DC C'THIS IS AN ERROR MESSAGE' MESSAGE TEXT
```

- An example showing two messages being sent. At the end of the first one (LINE1) there is no end of message character applied, while there is one applied at the end of the second (LINE2) due to the 0 in the rightmost byte of R4.

```
LA    R4,1
CWRTC MSG=LINE1,ENDOFMR=(R4)
SR    R4,R4
CWRTC MSG=LINE2,ENDOFMR=(R4)
```

- An example showing two messages being sent. On the first line (TESTMSG1) the time is replaced for 00.00.00 and the message code CSMP97I is appended on the front of the message (CSMP97I ABCD0001E ...). This assumes the proper SYSTC switch is set. The message is sent to the prime CRAS console. The second message consists only of the message text and this too is sent to the prime CRAS.

```
(5)    CWRTC MSG=TESTMSG1,TIME=YES,CSMP97I=YES,ROUT=(PRC)
        CWRTC MSG=TESTMSG2,TIME=NO,CSMP97I=NO,ROUT=(PRC)
        .
        .
        .
TESTMSG1 DC    AL1(L'MSG1TXT)
MSG1TXT  DC    C'ABCD0001E 00.00.00 ERROR IN TEST CASE 1'
TESTMSG2 DC    AL1(L'MSG2TXT)
MSG2TXT  DC    C'XYZ TABLE NOT INITIALIZED'
```

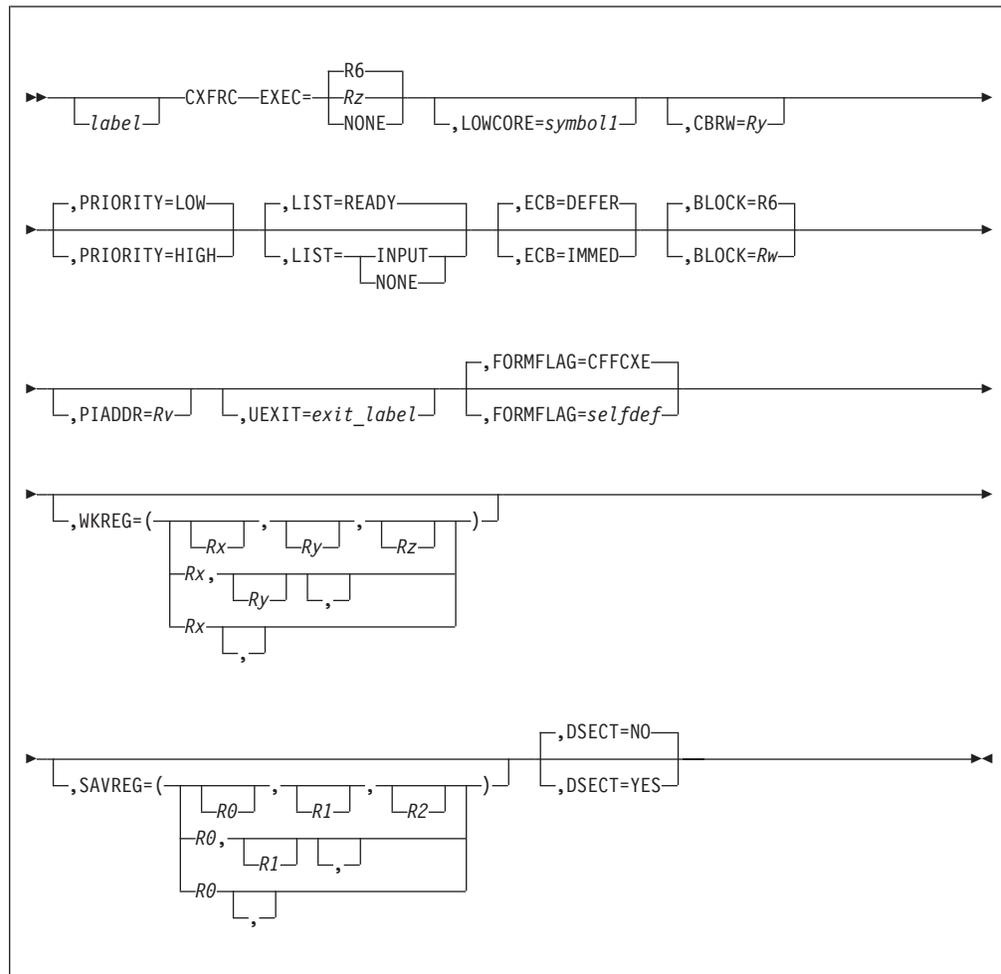
producing the message

```
CSMP97I  ABCD0001E xx.yy.zz ERROR IN TEST CASE 1'
        XYZ TABLE NOT INITIALIZED'
```

## CXFRC—Create a New ECB and Transfer Control

Use this system macro to create an entry control block (ECB) and permit the control program (CP) to transfer control to another program that is then free to use any CP macros.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### **EXEC=R6|Rz|NONE**

This parameter specifies the code (if any) to be processed after the ECB is dispatched. The parameter specifies a register that contains the address of 8 bytes of code to be processed (normally an enter expansion) or the value NONE, which indicates that the calling program will perform appropriate ECB formatting and scheduling). If the parameter is omitted, the default value is register R6. The NONE option is restricted to ECB=IMMED, LIST=NONE type calls.

#### **LOWCORE=symbol1**

Symbolic name of the user's routine for use in case of a low-core condition. If no routine is supplied, no check will be performed for the low-core condition. This parameter is only meaningful with DEFERred ECBs. See "Programming Considerations" on page 207 for more information.

**CBRW=Ry**

When no core block reference word (CBRW) is specified, the macro service routine will initialize CBRW 0 in the created ECB to empty.

When CBRW=Ry is specified, Rx designates a general purpose register which must point to an 8-byte field formatted as a CBRW. The block address in the CBRW must be a system virtual memory (SVM) address that will be valid on the target I-stream.

When CXFRC macro processing is complete, the 8-byte field pointed to by Ry will be formatted as an empty CBRW.

The CXFRC post interrupt will attach the storage at the SVM address to the address space of the created ECB (the ECB virtual memory), and initialize CBRW 0 with the data passed. The SVM address in the CBRW will be converted to the appropriate ECB virtual address.

**PRIORITY=LOW|HIGH**

Normally, a transfer block created to temporarily contain information relating to the ECB will be added to the bottom of the Ready List. This mode of processing reflects PRIORITY=LOW (ECB=DEFER, and LIST=READY), the defaults for each of these parameters. If the block (or ECB) is to be added to the top of the desired list, specify PRIORITY=HIGH.

**LIST**

The transfer block (or ECB) is added to the list specified for subsequent dispatching by the CPU loop.

**READY,**

Requests that the block be added to the Ready List.

**INPUT**

Requests that the block be added to the Input List.

**NONE**

Requests that the transfer block or ECB be created and its address be returned to the caller, and that the block is not to be added to any CPU loop lists at this time.

**ECB**

Specify one of the following:

**DEFER**

Specifies whether a transfer block is to be created to temporarily hold ECB information for later ECB creation, or whether the ECB is to be created immediately. The default is DEFER. If the ECB is created immediately, its address is returned to the caller in the register specified by the BLOCK parameter. If ECB creation is deferred, the address returned is that of the transfer block.

**IMMED**

Is restricted to code processing in the system virtual memory (SVM).

**BLOCK=R6|Rw**

Upon return from CXFRC macro processing, the register specified will contain either the address of the transfer block created to hold ECB information or the address of the ECB if the caller requested immediate ECB creation. The default value is register R6.

**PIADDR=Rv**

Upon return from CXFRC macro processing, the register specified will contain the address of CXFRC Post Interrupt Routine. If LIST=NONE was specified,

## CXFRC

this address may be used in a subsequent ADDxC macro call to request that the block being added to the desired list be processed by the appropriate Post Interrupt Routine. There is no default value for this parameter.

### **FORMFLAG=***selfdef*

A one-byte self-defining term may be supplied to be used to initialize the FORMAT FLAG field of the ECB. If the parameter is omitted, a default value of CFCXE will be provided by the TPF system.

### **UEXIT=***exit\_label*

This parameter specifies a label used in the CXFRC expansion to identify the origin of the control transfer. The label is used in the UCCCMXF user exit.

### **SAVREG**

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### **WKREG**

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

### **RTTMSG=***Ru*

The RTTMSG parameter passes the address of the message block to the Realtime Trace (RTT) interface.

There is no default. RTTMSG has no meaning when ECB=DEFER is used.

### **DSECT=**NO|YES

Optional parameter used to request the generation of a DSECT describing the parameter list generated for the CXFRC call. The use of this parameter should be limited to the CXFRC service routines. If DSECT=YES is coded, no CXFRC expansion is generated. The default is DSECT=NO.

## Entry Requirements

The EXEC parameter must specify a general purpose register that contains the address of 8 bytes of code (normally an enter expansion) that will be processed when the ECB is selected for processing.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Register contents will be preserved, except as noted below.
- If deferred ECB creation is requested, the BLOCK register contains the address of a transfer block which has bytes 8 through 114 (equivalent to the 107 bytes of the ECB work area) available. The complete format of this block is described by the ID SXFC DSECT. If LIST=NONE was not specified, this block is added to the requested CPU loop list. If this block is to be modified further, consideration

should be given to using the LIST=NONE option. The modified block can subsequently be added to the appropriate list through the use of one of the ADDxC macros.

- If immediate ECB creation is requested, the BLOCK register contains the address of the created ECB. If LIST=NONE was not specified the ECB is added to the requested CPU loop list. Similar consideration should be given as noted previously when the ECB is modified while residing on the requested list.
- If LIST=NONE was specified, the PIADDR=Rx parameter should be used to obtain the address of the appropriate Post Interrupt Routine which is to complete processing of the CXFRC request.

## Programming Considerations

- This macro may be processed on any I-stream.
- This macro is used only by the control program (CP) and certain test tool routines.
- ECBs that are created by the CXFRC macro will use the version of the program that was most recently activated. If this program is incompatible with the program that issued the CXFRC call, an interface problem may occur.
- Use of IMMED processing for ECBs is restricted to code processing in the system virtual memory (SVM).
- Use of EXEC=NONE is restricted to ECB=IMMED, LIST=NONE calls.
- If you have defined a lowcore routine, the number of available transfer blocks is compared against the number needed to service the CREDC macro. If the number available is less than the number specified on the LOWCORE parameter, a low-core situation is assumed and control relinquished to the user's error routine. If no lowcore routine is supplied, this processing is bypassed. The LOWCORE parameter has no meaning for IMMEDIATE ECBs processing.
- In addition to the normal macro trace information the macro trace entry contains the transfer type, IMMED or DEFER address of the transfer block and the associated format flag.
- If you use this macro to create an ECB that will enter a dynamic load module (DLM) with an entry point defined by the C language main function, the TPF system assumes that any core block attached to data level 0 (D0) contains a command string that will be parsed into argc and argv parameters for the main function. See *TPF Application Programming* for more information about the main function.

## Examples

- DSECT for the Parameter List:  
Requests a DSECT describing the parameter list for the CXFRC call. No code is generated.  
CXFRC DSECT=YES
- Simple Control Transfer:  
An example from the 37x5 interrupt handler shows passing control to a routine for a specific interrupt. The address for the routine to be transferred to (CS05\_CCIM) is specified in R6, the default register. ECB creation is deferred, so when the transfer returns, R6 contains the address of the transfer block (SWB) created and IDSXFC is a DSECT mapping this block. The transfer block will be added to the bottom of the Ready list. Registers R0-R2 are specified as being volatile and are saved on the stack.

## CXFRC

```

...
LA R6,CS05_CCIM GET ADDR OF 3705 MSG ROUTINE
CXFRC EXEC=R6,BLOCK=R6,
      PRIORITY=LOW,LIST=READY,ECB=DEFER,
      SAVREG=(R0,R1,R2)
IDSXFC REG=R6 Describe Transfer Block
...

```

- Transfer Call Using a Core Block Reference Word:

An example from a MPIF path error routine illustrates the use of the CBRW. Request an activation of a broadcast program (CBM6\_CBB0) specified in R14 with R3 pointing at a core block reference word formatted field (CBM6\_CR0). R4 is used to put the block type and size into a storage location (CBM6\_CT0) as a part of the field formatted as a core block reference word.

Upon return from the control transfer R14 contains the address of the transfer block created to hold ECB information, since ECB creation is deferred. When the ECB is activated, it is put on the bottom (low priority) of the Ready list. Register R1 is specified as a volatile register and is saved in the stack area. When the transfer returns, the IDSXFC DSECT is used to map the transfer block.

```

CBM6_CR0 DS F CBRW BLOCK ADDRESS (FOR CXFRC)
CBM6_CT0 DS F CBRW TYPE & SIZE (FOR CXFRC)
...
ST R4,CBM6_CR0 SET CBRW ADDRESS FOR CXFRC
LR R0,R15 SAVE BASE ACROSS MACROS
LA R4,L4 4K BLOCK TYPE
TYPBC TYPE=R4 GET BLOCK TYPE AND SIZE
LR R15,R0 RESTORE BASE REGISTER
ST R4,CBM6_CT0 SET TYPE & SIZE FOR CXFRC
L R4,CBM6_CR0 RELOAD BASE OF 4K BLOCK
LA R14,CBM6_CBB0 LOAD BASE OF BROADCAST
* ROUTINE
LA R3,CBM6_CR0 SET CBRW POINTER
CXFRC EXEC=R14,BLOCK=R14,CBRW=R3, PASS BLOCK TO CBB0
      PRIORITY=LOW,LIST=READY,ECB=DEFER,
      SAVREG=(R1)
IDSXFC REG=R14 DESCRIBE TRANSFER BLOCK
...

```

- Transfer Call with a LOWCORE Condition:

When this transfer block is taken off the Ready list, it will enter segment COSK. This transfer sets up a call to the program at CPMFRI and creates a deferred ECB. When the transfer is complete, R14 contains the address of the transfer block. If a lowcore condition is detected, control is transferred to the CPMPIND1 routine to either preserve a prior error code or indicate the current error code before returning.

```

LA R14,CPMFRI load COSK ENTNC address
CXFRC EXEC=R14, get transfer block for TZEC
      BLOCK=R14, function
      LOWCORE=CPMPIND1,
      PRIORITY=LOW,
      LIST=READY,
      ECB=DEFER,
      SAVREG=(R0,R1,R2)
SPACE 1
LA R15,CT7KEC load TZEC branch vector
...

***** CXFRC unable - set up WAITC error
SPACE 1
CPMPIND1 DS 0H
CLI CE1SUC(R9),X'00' previous error
BNE CPMPIND2 yes - do not clear SUG and SUD

```

```

SPACE 1
XC    CE1SUD(CE1SUG-CE1SUD+1,R9),CE1SUD(R9)  clear SUG / SUD
SPACE 1
MVI   CE1SUC(R9),X'30'          set I/O error flag
SPACE 1
***** return from macro
...

```

- Transfer Call Using Postinterrupt Address:

This transfer to routine CPMCPPI creates the ECB but since there is no list specified, the address of the ECB is returned in R14 and the address of the post interrupt routine is returned in R2. Observe that the ECB address is put into R9 for use as an ECB.

```

...
LA    R14,CPMCPPI          Locate 'ENTNC CPI'
CXFRC EXEC=R14,           Pass ENTNC addr in R14,
      BLOCK=R14,         Return ECB addr in R9,
      PIADDR=R2,        CXFRC PIA in R2,
      ECB=IMMED,        Make ECB now
      LIST=NONE,        But don't add to list
      SAVREG=(R0,R1)
LR    R9,R14              ECB address to R9
...

```

The value specified on the PIADDR parameter is used later as input to the \$ADPC macro, which adds the acquired ECB to the ready list. See “\$ADPC—Add Work to a List on Specified I-Stream” on page 12 for more information about the \$ADPC macro.



**DSECT**

Optional parameter that may be used to request the generation of a DSECT that describes the parameter list generated by the normal CYDNC macro expansion. The use of this parameter should be limited to the CYDNC service routines.

**YES**

A DSECT describing the parameters to be passed to the SVC service routine is generated.

**NO**

The normal macro expansion is generated (an SVC for IBM vectored SVC processing, the second level SVC (index) number, and parameters being passed to the SVC service routine). *The default is DSECT=NO.*

**Entry Requirements**

The register specified by the LIST= parameter must contain a CLH List value as noted above (for the CLES function only).

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- Register contents will be preserved.
- The register specified by the RETURN= parameter will contain the function results, as specified above.

**Programming Considerations**

- This macro may only be processed on the main I-stream.
- This macro is used only by the cycle down utility (currently limited to the CVCW segment).

**Examples**

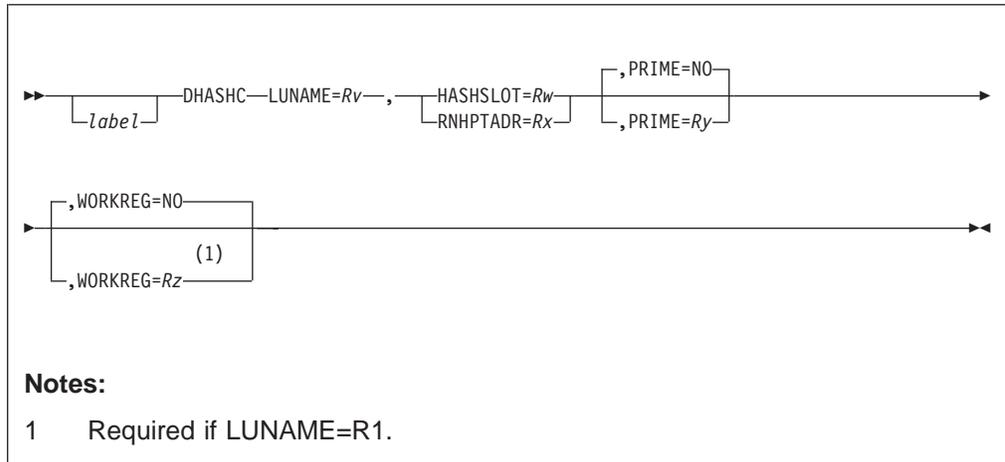
None.

## DHASHC–Hash Resource Name

Use this system macro to hash a resource name to obtain one or more of the following:

- Ordinal number of the node control block (NCB) directory record for the resource name
- Address of the resource name hash prime table (RNHPT) entry for the resource name.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **LUNAME=Rv**

A register (R1 through R7, or R14) that contains the address of a 16-byte resource name to hash.

#### **HASHSLOT=Rw**

A register (R1 through R7, or R14) that receives the ordinal number of the NCB directory record.

#### **RNHPTADR=Rx**

A register (R1 through R7, or R14) that receives the address of the RNHPT entry.

#### **PRIME**

A register that points to the fullword that contains the prime hash number.

#### **NO**

Use the largest prime number that is less than the value specified for the MAXPRIM parameter in the SNAKEY macro, which is the number of RNHPT entries defined in the TPF system. This is the default.

**Note:** Use the default value when you specify the RNHPTADR parameter.

*Ry* A register (R2 through R7) that contains the address of the prime hash number.

**Note:** If you specified the RNHPTADR parameter, the prime hash number cannot be greater than the largest prime number that is less than the value specified for the MAXPRIM parameter in the SNAKEY macro.

**WORKREG**

A register to preserve R1 across calls to the DHASHC macro. This parameter must be coded if LUNAME=R1 is specified.

**NO**

R1 is not preserved across calls to the DHASHC macro. This is the default.

*Rz* A register (R2 through R7) that preserves R1 across calls to the DHASHC macro.

**Entry Requirements**

R9 must contain the address of the requesting ECB.

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- R15 contains a return code, which indicates the following:
  - 0 Successful call.
  - 4 Value specified for the LUNAME parameter was not valid. The value cannot be all blanks or X'FFFFFFFFFFFFFFFF'.
  - 8 Hash value specified is greater than the allowed maximum value, which is the largest prime number that is less than the value specified for the MAXPRIM parameter in the SNAKEY macro.

**Note:** This condition can be returned only when you specify the RNHPTADR parameter. There is no maximum hash value defined for the HASHSLOT parameter.

**Programming Considerations**

- This macro can be run on any I-stream.
- On return, the contents of R0 and R1 are destroyed and R15 contains the return code.

**Note:** R1 is preserved if you specify the WORKREG parameter.

- The DHASHC macro does not verify that the specified resource actually exists. It simply returns the ordinal number of the NCB directory record or the address of the RNHPT entry for the resource, regardless of whether the resource exists or not.

**Examples**

None.

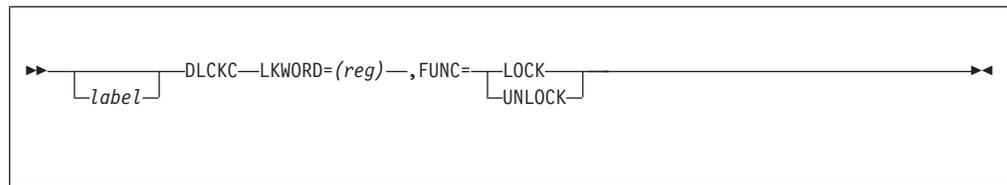
## DLCKC—Modify Lock and I/O Interrupt Status

Use this system macro to:

- Disable input/output (I/O) interrupts and lock a resource
- Unlock a resource and enable I/O interrupts.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X	X	

### Format



*label*

An optional label can be used with this macro.

**LKWORLD=(*reg*)**

A register containing the address of a doubleword lock field. R0 through R7 are valid registers. This parameter is required.

**FUNC**

Specify one of the following:

**LOCK**

The lockword specified by the LKWORLD parameter will be locked and I/O interrupts will be disabled on the I-stream issuing this macro.

**UNLOCK**

The lockword specified by the LKWORLD parameter will be unlocked and I/O interrupts will be enabled on the I-stream issuing this macro.

### Entry Requirements

- This macro should only be issued by ECB-controlled programs.
- The register referred to by the LKWORLD parameter must contain the address of a valid lockword name.

### Return Conditions

- Control is returned to the next sequential instruction.
- Registers R0 through R7 are saved. The contents of other registers cannot be predicted.

### Programming Considerations

This macro can only be used by ECB-controlled programs.

### Examples

- To disable I/O interrupts and lock the SSST in memory:

```
LA    R2, SSSTLCK
DLCKC LKWORLD=(R2), FUNC=LOCK
```

## DLCKC

This causes I/O interrupts on this I-stream to be disabled and the SSST to be locked. R2 points to the SSST lock (SSTHLCK).

- To enable I/O interrupts and unlock the SSST in memory:

```
LA    R5,SSTHLCK
DLCKC LKWORD=(R5),FUNC=UNLOCK
```

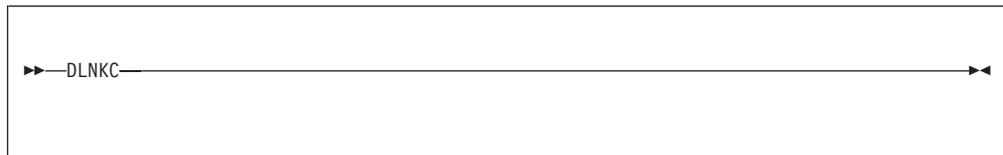
---

## DLNKC—Define Stack DSECT for Control Program (CP) Routine

Use this system macro to generate the stack DSECT for a routine that uses the standardized linkage macros. This macro is used with other standardized linkage macros such as the DLNKC, RLNKC, and SLNKC macros. See the following for more information about these macros:

- “DLNKC—Define Stack DSECT for Control Program (CP) Routine”
- “RLNKC—Return to CP Calling Routine and Reset Stack Pointer” on page 409
- “SLNKC—Control Program (CP) Save Link Data & Set Stack Pointer” on page 448.

### Format



### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- This macro must be used in conjunction with SLNKC to establish the stack DSECT where registers and data may be saved by this routine.
- Any data areas which reside in the stack for this routine must be defined between SLNKC and DLNKC.
- This macro can be run on any I-stream.
- Related macros are SLNKC, CLNKC, RLNKC.
- This macro is for use in the control program (CP) only.

### Examples

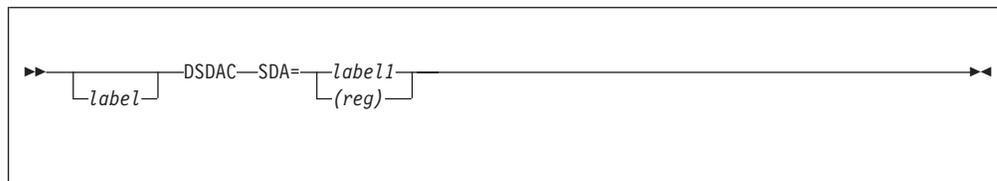
None.

## DSDAC—Dismount a Symbolic Device Address (SDA)

Use this system macro to perform a software dismount of a symbolic device address (SDA) that is associated with an input/output (I/O) device. Once this macro is processed for an SDA, the SDA is unavailable for any additional I/O until another MSDAC macro is issued for the SDA.

See “MSDAC—Mount a Symbolic Device Address (SDA)” on page 356 for more information about the MSDAC macro.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **SDA**

The SDA, which can be either of the following:

#### *label1*

A halfword field that contains an SDA.

#### *(reg)*

A register that contains an SDA in bytes 2–3 and zeros in bytes 0–1.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW key 0.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- One of the following return codes is returned in register 15 (R15):

Return Code	Meaning
0	The dismount is successful.
4	The dismount is not successful.

- The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can only be run on an I-stream having affinity with the specified SDA.
- This macro can be used only to dismount SDAs associated with I/O devices.
- This macro is for use in the control program (CP) only.

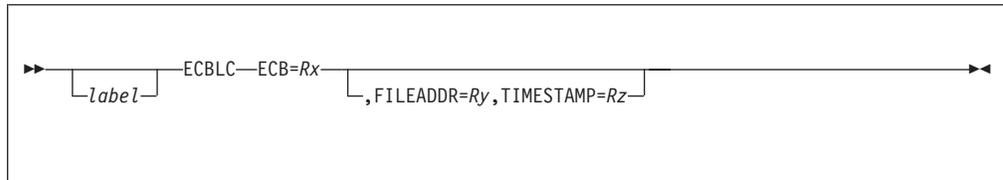
### Examples

None.

## ECBLC—Remove IOBs Associated with an ECB Address

Use this system macro to release input/output blocks (IOBs) associated with the specified entry control block (ECB). IOBs created by the macros that request file address holds will be found and removed from the related queue. The IOBs can be on the record hold table wait queue, the concurrency filter lock facility (CFLF) asynchronous queue, the limited lock facility (LLF) attention queue, or the coupling facility (CF) asynchronous queue.

### Format



#### *label*

specifies a symbolic name that can be assigned to the macro statement.

#### **ECB=R<sub>x</sub>**

specifies the system virtual memory (SVM) address of the ECB, where *R<sub>x</sub>* is a register from R0–R7.

#### **FILEADDR=R<sub>y</sub>**

specifies the pointer to a file address associated with a deadlock condition, where *R<sub>y</sub>* is a register from R0–R7.

#### **TIMESTAMP=R<sub>z</sub>**

specifies the pointer to a time stamp associated with a deadlocked file address, where *R<sub>z</sub>* is a register from R0–R7.

**Note:** *R<sub>x</sub>*, *R<sub>y</sub>*, and *R<sub>z</sub>* must be unique registers.

### Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- R14 will contain the return code:
  - 0 if IOBs were found and removed
  - 1 if no IOBs were found for the input ECB.
- The contents of R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

This macro can be run on any I-stream.

### Examples

```
ECBLC ECB=R3
```

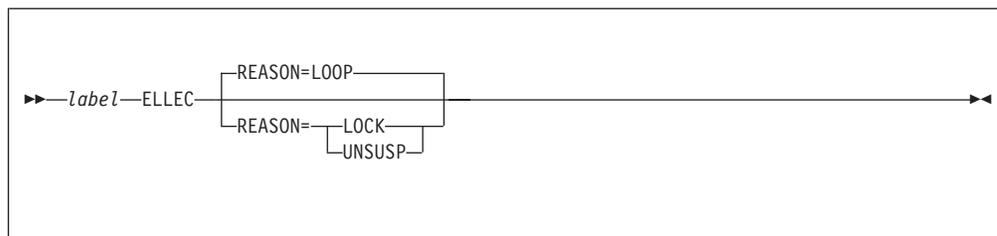
```
ECBLC ECB=R1,FILEADDR=R2,TIMESTAMP=R3
```

## ELLEC—Schedule an ECB to Exit or Resume

Use this system macro to schedule an entry control block (ECB) to exit or to resume a suspended ECB. The exiting ECB may be running or locked. This macro is used by the lock management program and by the ZECBL command with the E or U parameters specified.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### REASON

Specify one of the following:

##### LOOP

The ECB is being exited due to a ZECBL command request.

##### LOCK

The ECB is being exited because of a lock failure.

##### UNUSP

The suspended ECB is being dispatched to resume operation by a ZECBL command request.

### Entry Requirements

- R1 must contain the address of the ECB scheduled to exit or be dispatched to resume operation.
- R9 must contain the address of the ECB that is issuing the ELLEC macro.

### Return Conditions

- Control is returned to the instruction following the macro expansion.
- The contents of R14 and R15 cannot be predicted after you run the ELLEC macro.

### Programming Considerations

- The ELLEC macro is issued when the operator requests that an ECB be scheduled for exit or dispatched to resume operation (using the ZECBL command).
- The lock management program issues the ELLEC macro when a record ID that is being held by an ECB is being deleted.

## ELLEC

- If the entry that has been scheduled to exit regains control, the entry will be dumped and exited by the system error routines.

**Note:** R0 and R10 are modified by the routine used to send the entry to exit, and their contents, as shown in the system error dump, are not relevant.

- If the entry is not sent to exit one to 2 minutes after the ELLEC macro is run, it is flagged as *hung*, and a message is sent to the operator. No further attempt can be made to remove the entry from the TPF system.
- The ELLEC macro can be used on any I-stream.

## Examples

None.



## ENATC

### SAME

Inherits the C function trace setting most recently set by the ZSTRC command with the ALTER parameter specified and the CDEBUG/NOCDEBUG or XHOOKS/NOXHOOKS keywords specified for system-wide tracing or by the ENATC macro for individual entry control block (ECB) tracking. See *TPF Operations* for more information about the ZSTRC command.

See the *TPF Program Development Support Reference* for more information about C function trace.

## Entry Requirements

- For C-type code, code must be in key of zero and supervisor state when processing this macro, and R13 must point to a valid system stack area.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.
- The code issuing the ENATC macro must be on the same I-stream as the target ECB.
- The TPF system should be paused before issuing this macro on behalf of another ECB.

## Return Conditions

- Control is returned to the next sequential instruction.
  - For E-type code, the contents of R10 and R14 are unknown; R15 contains the return code.
  - For C-type code, register 15 contains the return code.
  - The contents of all other registers are preserved across this macro call.
  - Return codes in R15:
    - If no error occurs, R15 contains zero.
    - If an error occurs, the C function trace settings are not changed and an error indication code is returned in register 15. If there are multiple errors, control is returned to the caller after the first error is found.
- An error indication code is returned in register 15 when the ECB address in the register specified by the ECB parameter is not a valid system virtual address (SVA) ECB address.
- The following tags are generated by the ENATC macro and should be used for interrogating the error indication code:

IT_ENATC_BAD_CID_ERR	An incorrect SVM address of the C implementation data area (CID) of the target ECB was detected by the \$GSVAC macro. The C function trace settings are not changed.
IT_ENATC_BAD_CTTA_ERR	An incorrect SVM address of the C function trace table of the target ECB was detected by the \$GSVAC macro. The C function trace settings are not changed.
IIT_ENATC_BAD_ECB_ERR	The address in the ECB parameter register is not a valid system virtual address (SVA) ECB address. The C function trace settings are not changed.
IT_ENATC_BAD_TCA_ERR	An incorrect SVM address of the trace

communications area (TCA) of target ECB detected by \$GSVAC macro. The C function trace settings are not changed.

IT\_ENATC\_VA\_DEBUG\_ERR This call is not valid when the ECB is being debugged using TPF C Debugger for VisualAge Client.

- Trace hook settings on return.

At the completion of the macro, C function trace hook settings have been changed.

- An ENTRY parameter of:

**ON**

Changes the C function trace program entry breakpoint trace hook instruction for the specified ECB to branch to the C function trace program entry service routine.

A trace table entry is created on entry to a C function after the trace is started.

**OFF**

Changes the C function trace program entry breakpoint trace hook instruction for the specified ECB to a no-operation instruction.

A trace table entry is not created on entry to a C function.

**SAME**

Does not change the C function trace program entry breakpoint trace hook instruction for the specified ECB, but retains its current setting.

- An EXIT parameter of:

**ON**

Changes the C function trace program exit breakpoint trace hook instruction for the specified ECB to branch to the C function trace program exit service routine.

A trace table entry is created on exit from a C function.

**OFF**

Changes the C function trace program exit breakpoint trace hook instruction for the specified ECB to a no-operation instruction.

A trace table entry is not created on exit from a C function.

**SAME**

Does not change the C function trace program exit breakpoint trace hook instruction for the specified ECB, but retains its current setting.

- An OTHERS parameter of:

**ON**

Changes the other C function trace breakpoint trace hook instructions for the specified ECB to branch to the C function trace other breakpoints service routine.

A trace table entry is created for the breakpoint when the breakpoint instruction is invoked after the trace is started.

The program entry breakpoint trace hook instruction and the program exit breakpoint trace hook instruction for the specified ECB are *not* changed by ENATC OTHERS=ON. All other trace hook instructions for the specified ECB are changed.

## ENATC

### OFF

Changes the other C function trace breakpoint trace hook instructions for the specified ECB to a no-operation instruction.

A trace table entry is not created on the invocation of any of the other breakpoints.

The program entry breakpoint trace hook instruction and the program exit breakpoint trace hook instruction for the specified ECB are *not* changed by ENATC OTHERS=OFF. All other trace hook instructions for the specified ECB are changed.

### SAME

Inherits the C function trace setting most recently set by the ZSTRC command with the ALTER parameter specified and the CDEBUG/NOCDEBUG or XHOOKS/NOXHOOKS keywords specified for system-wide tracing or by the ENATC macro for individual entry control block (ECB) tracking. See *TPF Operations* for more information about the ZSTRC command.

The program entry breakpoint trace hook instruction and the program exit breakpoint trace hook instruction for the specified ECB are *not* changed by ENATC OTHERS=SAME.

## Programming Considerations

- When an ECB is created, it inherits the system-wide C function trace setting. The ENATC macro allows you to enable or disable C function trace for program entry, program exit, and other breakpoint instructions for an individual ECB.
- Code ENTRY=SAME, EXIT=SAME, and OTHERS=ON when you want to enable C function trace for extra or other breakpoints, but do not want to change the current setting of the program entry or program exit breakpoints.  
You should code OTHERS=ON or OTHERS=OFF rather than OTHERS=SAME because OTHERS=SAME is dependent on inherited conditions while OTHERS=ON and OTHERS=OFF are independent of inherited conditions.
- Coding OTHERS=SAME inherits the C function trace setting most recently set by the ZSTRC command with the ALTER parameter specified and the CDEBUG/NOCDEBUG or XHOOKS/NOXHOOKS keywords specified for system-wide tracking or by the ENATC macro for individual ECB tracking.

For example:

- Coding OTHERS=SAME disables the C function trace of breakpoints other than program entry breakpoints and program exit breakpoints for the specified ECB when:
  - The ZSTRC command is entered with the ALTER parameter and the NOHOOKS keyword specified, system-wide tracking of extra or other hooks is disabled
  - OTHERS=OFF was coded previously, ECB tracing of extra or other hooks for an individual ECB is disabled.
- Coding OTHERS=SAME enables C function trace of breakpoints other than program entry breakpoints and program exit breakpoints for the specified ECB when:
  - The ZSTRC command is entered with the ALTER parameter and the CDEBUG and XHOOKS keywords specified, system-wide tracing of extra or other hooks is enabled
  - OTHERS=ON was coded previously, ECB tracing of extra or other hooks for an individual ECB is enabled.

- If all of the ENTRY, EXIT, and OTHERS parameters are coded with the SAME parameter, a macro error message occurs because the macro effectively becomes a no-operation (you have not modified any breakpoint).
- The CTRC user exit is provided for you to trace additional information.
- Use the ENATC macro to trace ISO-C programs that have been compiled using the TEST option of one of the IBM C/370 family of compilers supported by the TPF 4.1 system.

## Examples

- ENATC ECB=R6,ENTRY=ON,EXIT=ON,OTHERS=OFF

This invocation:

- Changes the C function trace for the ECB whose SVA is contained in register 6.
- Enables the C function trace at function entry point.
- Enables the C function trace at function exit point.
- Disables the C function trace breakpoints for the BLOCK, LINE, PATH, and so on.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- ENATC ECB=R5,ENTRY=OFF,EXIT=SAME,OTHERS=ON

This invocation:

- Changes the C function trace for the ECB whose SVA is contained in register 5.
- Disables the C function trace at function entry point.
- Does not change C function trace at the function exit point.
- Enables C function trace breakpoints for the BLOCK, LINE, PATH, and so on.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- ENATC ECB=R4,ENTRY=OFF,EXIT=OFF,OTHERS=ON

This invocation:

- Changes the C function trace for the ECB whose SVA is contained in register 4.
- Disables the C function trace at the function entry point.
- Disables the C function trace at the function exit point.
- Enables the C function trace breakpoints for BLOCK, LINE, PATH, and so on.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

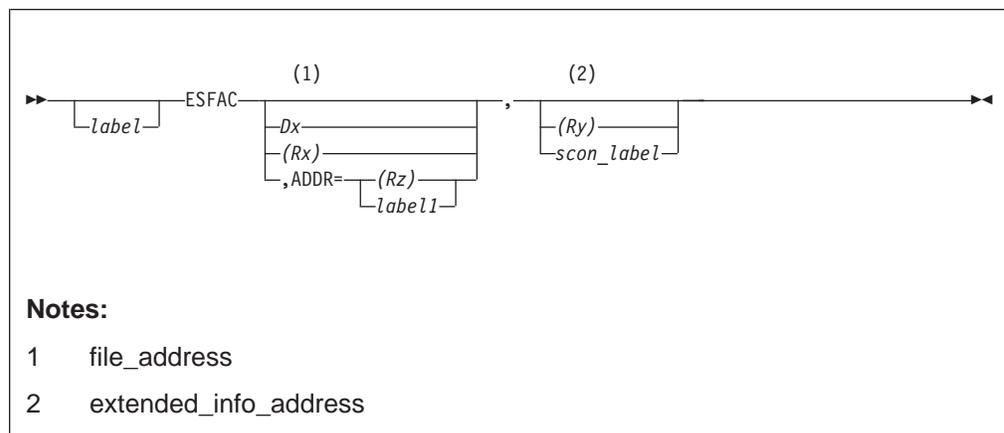
## ESFAC—Obtain Symbolic File Address Information

Use this system macro to obtain information about the characteristics of a specific symbolic file address as follows:

- The address is a fixed or pool address.
- The record is small, large or 4 K in size.
- The record is duplicated or nonduplicated.
- For a pool file address reference format (FARF) address, the address is short-term or long-term.
- If a fixed FARF address is common to all processors, I-streams, and users of the subsystem, or is unique to a combination of processors, I-streams, and users of the subsystem.
- If the address is a FARF3, FARF4, FARF5, or FARF6 address.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### file\_address

Location of the file address.

*Dx* A file address reference word (D0–DF) which contains the file address.

*(Rx)*

A register that contains the file address. For E-type programs this may be R0–R7. For control programs (CPs), this may be R1–R10, R14, or R15.

**ADDR=(Rz)|label1**

This parameter specifies the general register containing the location, or a label indicating the location, of an 8-byte file address. For E-type programs, this can be R1–R7. For CPs, this can be R1–R10, R14, or R15.

#### extended\_info\_address

This parameter specifies the address of where the extended information is to be returned.

*(Ry)*

A register specifying the address where the extended information is to be returned. For E-type programs this may be R1–R7. For CPs, this may be R0–R10, R14, or R15.

*scon\_label*

A label that is resolvable as an S-type constant. For E-type programs, the SCON base must be in the range R1–R7 or R9 if an entry control block (ECB) work area is used. For CPs, the SCON base may be any register.

The address specified by the second positional parameter points to a data area structured according to DSECT DCTSON.

## Entry Requirements

- If an ECB data level is specified to hold the file address, R9 must contain the address of an ECB.
- The symbolic file address must be contained in file address reference word (FARW) CE1FM $x$ , where  $x$  is the specified ECB data level or it must be contained in a register. If this information is not provided, the ADDR parameter must be coded to point to an 8-byte file address.

## Return Conditions

- Control is returned to the next sequential instruction.
- Condition code 0 is returned if the file address is successfully decoded; condition code 1 if it is not.
- The FARW at the specified level, if used, is unchanged.
- For E-type programs, R0–R7 are saved in the ECB at CE1SVR and restored on return from the service routine. For CP programs, all registers are preserved across the macro call except R0 and R1 that are used for parameters and R14 and R15 that are used for linkage to the service routine.

## Programming Considerations

This macro can be run on any I-stream.

## Examples

None.

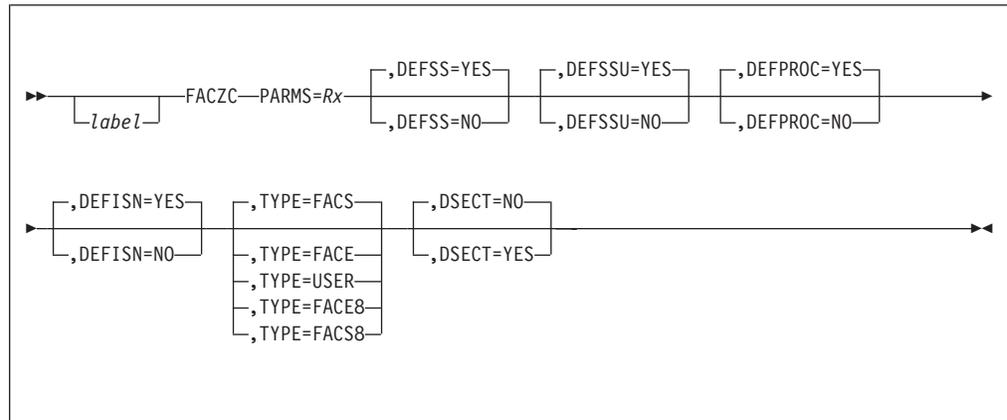
## FACZC—Compute File Address

Use this system macro to provide the interface to file address compute program (FACE) address generation routines. The service is similar to calling the FACS segments or the FAC8C macro service routine, but additionally this service allows access to records that are unique to a subsystem user (SSU), processor, or I-stream (within a given subsystem).

If defaults are taken, an entry control block (ECB) must be available because the subsystem, DBI, and I-stream of the current ECB are used. Control program (CP) routines with no associated ECB can take defaults for only the processor and I-stream, and must provide subsystem and SSU information.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



*label*

A symbolic name may be assigned to the macro statement.

### PARMS=Rx

Rx specifies a register (R0 through R7 for E-type segments, any register except R0, R11 through R13 for CP segments) containing the starting address of a parameter block, as described by the IDSFCZ macro. Fields in the input area of this block that are to be provided by the caller must be initialized before invocation of the macro. Output area fields will be filled in by the macro service routine. In particular, the return code field should be inspected to determine the results of the call.

**Note:** The IFCZREC field must contain an 8-byte character string record type name, padded on the right with blanks.

### DEFSS

Specify one of the following:

#### YES

Sets a parameter block value so that the current DBI is used to find the FACE table. The caller must ensure an ECB address is contained in R9.

**NO**

Indicates that the correct subsystem index will be provided in the parameter block by the caller.

**DEFSSU**

Specify one of the following:

**YES**

Sets the parameter block value for SSU to the current SSU from the ECB. This value is then used to locate the proper record for fixed record types for which any SSU/processor/I-stream uniqueness exists. The caller must ensure an ECB address is contained in R9.

**NO**

Indicates that the correct subsystem user index will be provided in the parameter block by the caller.

**DEFPROC**

Specify one of the following:

**YES**

Sets the parameter block value for processor to the current processor ordinal from the TPF system. This (zero-relative) value is then used to locate the proper record for fixed record types for which any SSU/processor/I-stream uniqueness exists.

**NO**

Indicates that the processor ordinal will be provided in the parameter block by the caller.

**DEFISN**

Specify one of the following:

**YES**

Sets the parameter block value for I-stream to the current I-stream number from the ECB, if an E-type call otherwise the current I-stream number is used. This (one-relative) value is then used to locate the proper record for fixed record types for which any SSU/processor/I-stream uniqueness exists. The caller must ensure an ECB address is contained in R9.

**NO**

Indicates that the I-stream number will be provided in the parameter block by the caller.

**TYPE**

Specify one of the following:

**FACE**

Sets a parameter block value to indicate a FACE-type call. The record type number must be placed in the IDSFCZ parameter block input area prior to macro invocation.

**FACS**

Sets a parameter block value to indicate a FACS-type call. The symbolic record type name must be placed in the IDSFCZ parameter block input area prior to macro invocation. The default value is TYPE=FACS.

**USER**

The appropriate entry type information must be set prior to calling FACZC. The entry type flag must be set. The record type field must be initialized to either a 8-character symbolic record type name (FACS interface) or a 2-byte record type number (FACE interface).

## FACZC

### FACE8

Sets a parameter block value to indicate an 8-byte file address type of call. An 8-byte ordinal number and record type number must be supplied on input, and an 8-byte file address and an 8-byte maximum ordinal number are provided on output in the IDSFCZ parameter block. This call only supports the record-type number interface (similar to the FACE interface).

### FACS8

Sets a parameter block value to indicate an 8-byte file address type of call. An 8-byte ordinal number and the symbolic type name must be supplied on input, and an 8-byte file address and an 8-byte maximum ordinal number are provided on output in the IDSFCZ parameter block. This call only supports the symbolic record-type interface (similar to the FACS interface).

### DSECT=NO|YES

Optional parameter used to request the generation of a DSECT describing the parameter list generated for the FACZC call rather than the generation of service routine linkage. The use of this parameter should be limited to the FACZC service routine. If DSECT=YES is coded, no FACZC expansion will be generated. The default is DSECT=NO.

## Entry Requirements

- PARMS=Rx must specify a register containing the address of a block of contiguous, addressable storage as described by the IDSFCZ macro. These input area fields must be filled in with correct values or the default must be specified on the FACZC macro.  
The ordinal requested and either symbolic record type or record type number fields are required (depending on the FACE or FACS interface chosen). All other fields are optional, except for CP routines with no associated ECB. These routines must supply all the SSI and SSU values, since there is no ECB from which to take default values.
- If these values are to be defaulted, an ECB must be addressable in the calling routine. To provide one of these values, DEFxxxx=NO must be coded, in addition to field initialization before the call.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- For ECB-controlled segments, the contents of registers R0 through R7 will be preserved. The contents of R14, and R15 are unpredictable.
- For unique records the IFCZUNIQ bit in the IFZCUNIQ field of DSECT IDSFCZ is set. Bits IFCZPID, IFCZIID, IFCZSID, and IFCZUND in field IFCZUNQ of IDSFCZ are set indicating the owning processor, I-stream, ssu and uniqueness type respectively.
- For CP segments, the contents of registers R1 through R13 and R15 will be preserved; R0 will be used for parameter passing; the contents of R14 is unpredictable.
- Output area fields will be filled in by the macro service routine, as specified in the IDSFCZ definition.
- The FARFx file address will be returned if the return code in the parameter block indicates a normal return.
- If TYPE=FACE8 or TYPE=FACS8, an 8-byte file address and an 8-byte maximum ordinal number are returned.

The following return conditions are found in the IFCZRET field.

Field Name	Condition Value	Description of Error
IFCZNRM	0	Normal Return
IFCZNIU	1	Record Type is Not In Use
IFCZRTH	2	Record Type does not exist or exceeds FACE table limit
IFCZROR	3	Record ordinal number is outside allowable range
IFCZNSP	4	No split chain for record
IFCZPOR	5	Input parameter is outside of allowable range.

- Field IFCZMNX contains the largest ordinal number for the referenced record type. If IFCZMNX is zero, an error occurred that is reported in IFCZRET. If TYPE=FACE8 or if FACS8 was used for this call, the maximum 8-byte ordinal number is in field IFCZMX8. If IFCZMX8 is zero, an error occurred.
- The FARF address for that record type and ordinal (for the subsystem and SSU, processor, I-stream) will be returned, unless an error has occurred.
- If the return code indicates an error, it will also specify what type of error occurred.
- If the error is an ordinal range error, the next available ordinal number will be returned, unless the requested ordinal is higher than the record type allows, in which case zero is returned.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro may be used by E-type programs or control program (CP) code.
- If used by CP routines which do not have access to an ECB, the SSUI and SSU values must be provided. This is done by initializing the input fields of the IDSFCZ parameter block before the FACZC macro is issued.
- A symbolic record type (FACS interface) or record type number (FACE interface) and an ordinal in that record type must be provided. The type of call (FACS, the default, FACE, FACE8, or FACS8) is specified using the TYPE parameter. Optionally, a subsystem, subsystem user, processor, and I-stream (or any combination of these) can be provided.
- Pool FARF addresses can be generated just like fixed file addresses by using FACZC as well as FACE and FACS. The following record type names are predefined for the various pool types.

Pool Type	Description
#IPSLT	Small Long-Term Non-Duplicated
#IPSST	Small Short-Term
#IPSDP	Small Long-Term Duplicated
#IPLLT	Large Long-Term Non-Duplicated
#IPLST	Large Short-Term
#IPLDP	Large Long-Term Duplicated
#IP4LT	4 K Long-Term Non-Duplicated

## FACZC

#IP4ST            4 K Short-Term  
#IP4DP            4 K Long-Term Duplicated.  
#IP4D6            4 K Long-Term Duplicated FARF6.

This interface is not a substitute for the GETFC interface. Pool file addresses must be obtained initially using the GETFC macro. See *TPF General Macros* for more information about the GETFC macro.

## Examples

The following is an example of a FACZC macro call.

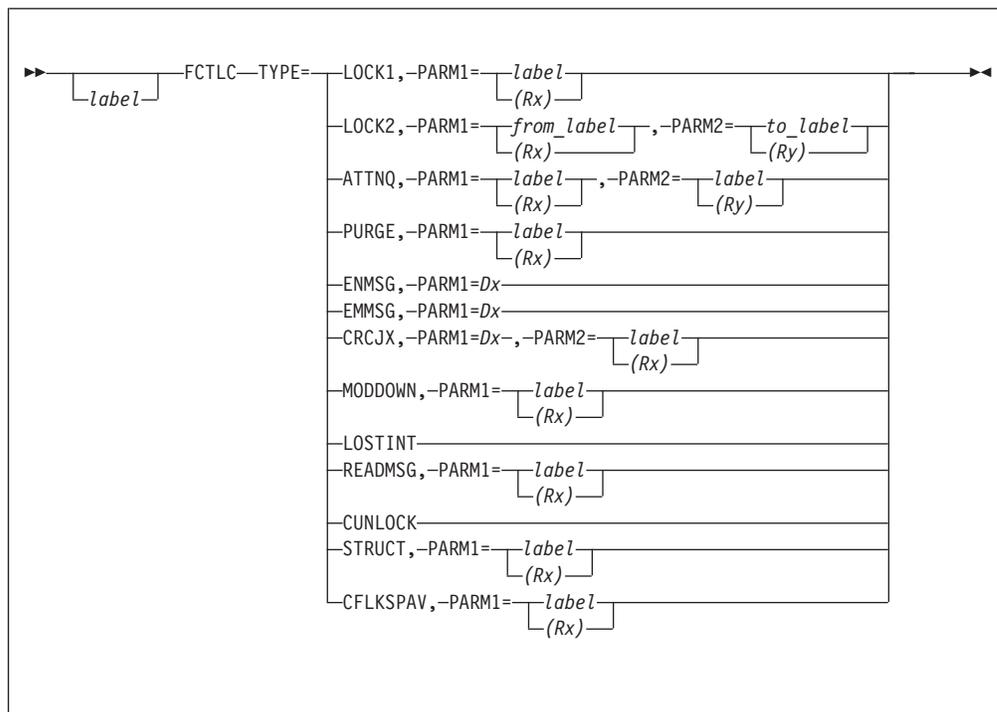
```
XC   IFCZORD,IFCZORD   ORDINAL ZERO
MVC  IFCZREC(8),=CL8'#PDREU'  FACE TYPE
XC   IFCZPRO,IFCZPRO   CLEAR OUT
MVC  IFCZPRO+1(1),PI1IPT SET UP PROCESSOR ORDINAL
FACZC PARS=R5,        GET FILE ADDRESS
      DEFSS=YES,      USING DEFAULT SS
      DEFSSU=YES,     SSU
      DEFISN=YES,     I-STREAM
      DEFPROC=NO,
      TYPE=FACS,      USING FACS INTERFACE
      DSECT=NO
CLI  IFCZRET,IFCZNRM   OK RETURN?
BNE  CBRIERR1         NO, ERROR
L    R2,IFCZMNX       SAVE MAXIMUM RECORD
MVC  CE1FM2(4),IFCZADR MOVE ADDRESS TO LEVEL 2
XC   CE1FA2(4),CE1FA2  CLEAR ID, RCC
FIWHC D2,CBRIERR2
```

## FCTLC–File Control

Use this system macro to request control program (CP) DASD services from the CCSONS CP segment. This macro is restricted to specific use by the DASD support routines because the services provided are a code continuation of the issuing real-time program.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



*label*

A symbolic name can be assigned to the macro statement.

### TYPE

The name of the process to be supplied by CCSONS. The values of PARM1 and PARM2, if used, depend on the service being requested. TYPE is one of the following:

#### LOCK1

Lock phase 1 processing for move locks support. This causes CCSONS to move any hold-type requests from the module queue to the module control unit attention queue. If a hold-type request is active, the request is halted.

#### PARM1

The label of a halfword that contains the relative module number or a register value in the format (Rx) that contains the relative module number. The MFST entry of the subject module must be marked in LOCK SUSPENDED state.

**LOCK2**

Lock phase 2 processing for move locks support. This causes CCSONS to move any requests on attention queue of the FROM module to the attention queue of the TO module.

**PARM1**

The relative module number of the FROM module. This must be specified as either the label of a halfword that contains the relative module number or a register value in the format (*Rx*) which contains the relative module number. The MFST entry of the subject module must be marked in LOCK SUSPENDED state.

**PARM2**

The relative module number of the TO module. This must be specified as either the label of a halfword that contains the relative module number or a register value in the format (*Ry*) which contains the relative module number. Register 14 cannot be used for PARM2 value. The MFST entry of the subject module must be marked in LOCK SUSPENDED state.

**ATTNQ**

Process the attention queue and attempt to schedule all pending requests.

**PARM1**

Either the label of a halfword that contains the relative module number or a register value in the format (*Rx*) that contains the relative module number.

**PARM2**

Specify the relative module number of the target module. Specifying a -1 indicates a lock recovery request for a CFLF control unit (CU). The specification must be either the label of a halfword or a register (other than R14) containing the module number.

**PURGE**

Stop any active request and purge the rest of the device queue for the specified module.

**PARM1**

Either the label of a halfword that contains the relative module number or a register value in the format (*Rx*) that contains the relative module number. The MFST entry of the subject module must be marked as unavailable to normal system requests.

**ENMSG**

Call CJIW to create the appropriate text for a module down message based on the information provided by the input work block (ICJXWK).

On return, the ICJXWK block contains the message at CJXWKMSG.

**PARM1**

The data level index that contains the initialized ICJXWK block. PARM1 is specified as a character string of *Dx*, where *x* is the hexadecimal data level number (0–F).

**EMMSG**

Call CJIW to create the appropriate text for a device error message based on the information provided by the input work block (ICJXWK).

On return, the ICJXWK block contains the message at CJXWKMSG.

**PARM1**

The data level index that contains the initialized ICJXWK block. PARM1 is specified as a character string of *Dx*, where *x* is the hexadecimal data level number (0–F).

**CRCJX**

Allocate and initialize an ICJXWK block to use for error recording and logging.

On return, an initialized ICJXWK block is attached to the data level specified by PARM1 for the module specified by PARM2. The address of the block is at CE1CR*x*, where *x* is the data level (0–F).

**PARM1**

An available data level index on which the initialized ICJXWK block will be returned. PARM1 is specified as a character string of *Dx*, where *x* is the hexadecimal data level number (0 through F).

**PARM2**

Either the label of a halfword that contains the relative module number or a register value in the format (*Rx*) which contains the relative module number of the module whose block is being initialized.

**MODDOWN**

Force the specified module offline.

**PARM1**

The label of a halfword that contains the relative module number or a register value in the format (*Rx*) that contains the relative module number of the module that is to be forced down.

**LOSTINT**

Scan for lost interrupts.

**READMSG**

Request a Read Attention I/O operation to the specified device.

**PARM1**

The label of a halfword that contains the relative module number or a register value in the format (*Rx*) that contains the relative module number of the module that is to be used to perform the read attention message I/O operation.

**CUNLOCK**

Issue any cached unlocks.

**STRUCT**

Process the coupling facility (CF) list structure queue and attempt to schedule all pending requests.

**PARM1**

Either the label of a halfword that contains the relative module number or a register value in the format (*Rx*) that contains the relative module number.

**CFLKSPAV**

Process the CF list structure queue used by CF record lock support to reschedule all pending requests that were queued because of a lock table full condition.

**PARM1**

The label of a halfword or register value in the format (*Rx*) that contains the index (0–31) in the CF locking table (CFLT) for the CF whose

## FCTL

structure queue must be examined so that the input/output blocks (IOBs) can be restarted if they were queued because of a lock space full condition.

### Entry Requirements

R9 must contain the address of the ECB that is being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of registers R0–R7 are preserved across this macro call.
- The condition code on return from this macro is unknown.

### Programming Considerations

- This macro can be run on the main I-stream only.
- To ensure completion of the operation, a WAITC macro must be issued.
- Only system programmers should use this macro. This macro is designed to be an extension of real-time code sequences.

### Examples

None.

## FDCTC–File Data Chain Transfer

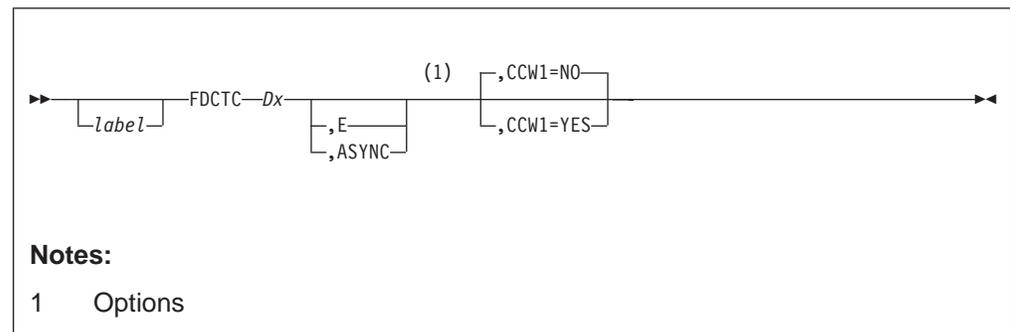
Use this system macro to process a chain of user-defined format-0 or format-1 channel command words (CCWs) to:

- Read records from a file into processor storage
- Write records from processor storage to a file.

This is a single instance FIND or FILE macro. Only the copy of the record specified by the user-defined address will be read or written regardless of the record's duplication factor. This macro does not obtain storage from or return storage to any storage pool.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### *label*

A symbolic name may be assigned to the macro statement.

*Dx* A file address reference word (D0-DF) must be specified.

### *options*

The following options are available:

**E** Extended status and error data are requested.

### **ASYNC**

Requests that the I/O operation receive asynchronous notification of completion for operations that may take a long time to complete. Operations in this category are associated with the Perform Subsystem Function and Set Subsystem Mode CCW.

### **CCW1=NO|YES**

This parameter is used to indicate the CCW format in the specified data level. If `CCW1=YES`, the CCW is treated as CCW format-1, otherwise CCW format-0 is assumed. *CCW1=NO is the default.*

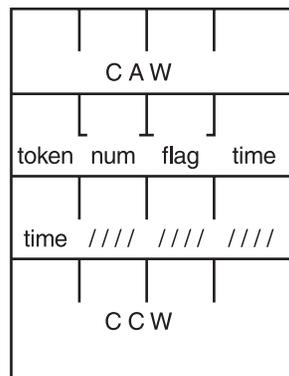
## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A module number and the storage location of the first user-supplied CCW must be contained in the file address reference word (FARW) for the specified data level. The format follows here.

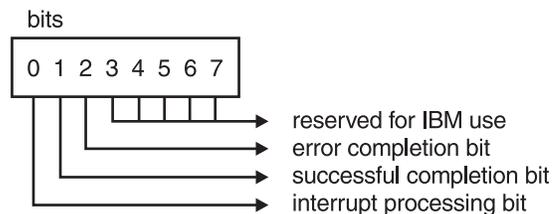
# FDCTC

CE1FAx	Bytes 0–3	Storage address of first CCW.
CE1FMx	Bytes 0–1	2-byte module number (general data sets use byte 0 only).

- You must specify all fully constructed chained CCWs desired, including the SEEK command, before processing of this macro.
- Storage locations into which data is to be read to or written from must be obtained by you and inserted into the CCWs before processing of this macro.
- If E is specified, the core block reference word, CE1CRx, associated with the FARW, must contain a storage block formatted using the FI0CB DSECT:
  - The channel address word (CAW) stored in FI0CAW. The storage key must be in the high order byte of the FI0CAW.
  - The FI0FLG set so that:
    - Bit 0 = 0      SONS will retry the error.
    - Bit 0 = 1      SONS will not retry the error.
    - Bit 1 = 0      Return sense data from the last retry.
    - Bit 1 = 1      Return sense data from the first retry.
    - Bits 2 through 7      Reserved for use by IBM.
  - FI0PATH field indicates the user specified path for SONS to use. If FI0PATH is 0, SONS will use the logical path mask indicated in the MFST (MF1LPM).
  - Minimum size 40 hexadecimal bytes (64 decimal) reserved.
- If ASYNC is specified, the core block reference word, CE1CRx, associated with the FARW (FI0CB), must contain a core block formatted as follows.



- The channel address word (CAW) stored at location FI0CAW
- The asynchronous event token number stored at location FI0TKN
- The flag byte at location FI0AIND, set so that



- Bit 0 = 0      The asynchronous event lost interrupt processing is allowed.
- Bit 0 = 1      The asynchronous event lost interrupt processing is inhibited.

- Bit 1 = 0      The attention message buffer is kept on successful completion.
- Bit 1 = 1      The attention message buffer is discarded on successful completion.
- Bit 2 = 0      The attention message buffer is kept on error completion.
- Bit 2 = 1      The attention message buffer is discarded on error completion.

Bits 3 through 7

Reserved for use by IBM.

- The asynchronous event timeout value (in seconds) should be stored at location FIOTIME if a long event is expected. The default timeout value is 5 minutes.

FI0LIT    DS    H                    ASYNC LOST INTERRUPT TIMEOUT VALUE

FIOTIME   EQU   (5\*60)            DEFAULT TIMEOUT VALUE (5 MINUTES)

- The CCW chain must contain a single Set Subsystem Mode or Perform Subsystem Function CCW.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- If this macro is called from a program running in 24-bit mode, the condition code is saved across this macro call. If the macro is called from a program running in 31-bit mode, the condition code upon return from this macro is unknown.
- The status of the operation is unknown, unless E or ASYNC is specified.
- The contents of the core block reference word (CBRW) at the specified level is unchanged, unless E or ASYNC is specified.
- The FARW at the specified level is unchanged, on a normal return. On an error condition, the address portion of the channel status word (CSW) is saved at CE1FAx, unless E or ASYNC is specified.
- If E or ASYNC is specified and no error occurs, the CSW is returned at location 8 in the area pointed to by CE1CRx.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is restricted to system use only. Use caution when using this macro, as it is possible to destroy data on the random access files.
- No check of record type or record code check is made.
- The operational program may not use the specified level upon return from the control program (CP).
- To ensure completion of the operation, a WAITC macro must be issued.
- The control program (CP) will:
  - Translate CKD CCWs to ECKD (\*) CCWs if the device is buffered or cached and the E or ASYNC option is not specified
  - Transfer control to the user-supplied CCWs
  - Handle the following error conditions:
    - Device not ready or intervention required

## FDCTC

- Defective track condition
- Seek check or data check
- No record found
- Unit exception.
- Upon noncorrectable error conditions, the gross error indicator for the specified level will be set, and the detailed error indicator's bit zero will be set to 1. Also, the address portion of the CSW will be inserted into the first 4 bytes of the FARW for the specified level, unless E or ASYNC is specified.
- The CCW prefetch bit in the operation request block (ORB) is **not** enabled for FDCTC channel programs.

## Examples

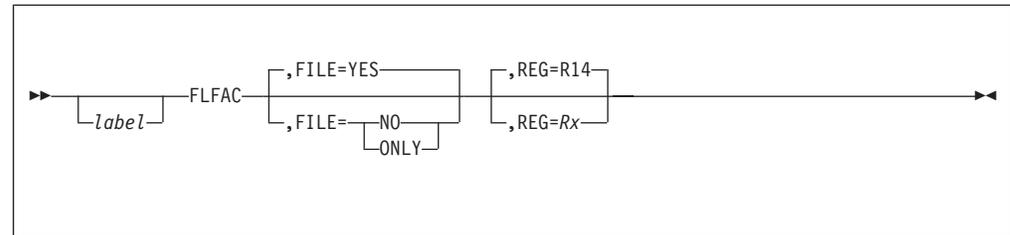
None.

## FLFAC—Flush a Record from VFA Buffers

Use this system macro to flush a record from the virtual file access (VFA) buffers. The next request for this record will force a read from file.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### FILE

Specify one of the following:

#### YES

Requests the flushed record be filed out to DASD if the record is delayed pending a file. YES option is assumed when the FILE parameter is omitted.

#### NO

Indicates the record should not be filed out to DASD, even if it is delayed file pending.

#### ONLY

The flushed record is filed out to DASD *only* if it is delayed file pending. If it is not delayed file pending, no change is to be made to the record's VFA residence (and it is available in VFA).

### REG=R14|Rx

A general register, one of R0 through R7 or R14 through R15, containing the address of a parameter list consisting of:

- FARF address of the record to be flushed. (4 bytes)
- DBI and complement of record to be flushed. (2 bytes)

**Note:** If issuing this on a program record, the PBI and complement of the issuing ECB should be used.

- SSU and complement of record to be flushed. (2 bytes)
- File byte indicator (1 byte set up by the macro itself)

When the REG parameter is omitted, R14 is assumed.

## Entry Requirements

- This macro can only be called from E-type programs.

## FLFAC

- The routine calling the FLFAC macro must set up the file address reference format (FARF) address, DBI and complement, and subsystem user (SSU) and complement, as well as provide space for the file byte indicator.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R0 through R7 are preserved across this macro call.
- The macro service routine will set R14 to contain a hexadecimal return code that can be used by the calling program.

R14 = 00 → Record was successfully flushed from VFA

R14 = 04 → Record not in VFA

R14 = 08 → Record found but was not filed

R14 = 0C → Invalid file address

- The contents of R15 are unknown.

## Programming Considerations

- This macro can be run on any I-stream.
- When issuing a FLFAC on a program record the PBI and compliment should be used rather than the DBI and compliment. The record was brought into VFA using the PBI and compliment to access the correct control tables so the same should be used to flush the program record.
- If the program to be flushed has been locked in memory (using the GETPC macro), the FLFAC macro will not flush the program until it is unlocked (by using the RELPC macro).
- If a FLFAC macro is issued within a commit scope, the effect of the FLFAC macro is deferred until the root scope commits the transaction.

The processing sequence of file-type macros and FLFAC macros is important. In a commit scope, the FLFAC macro preserves the logical sequence of files and FLFACs. The following sequence results in a value of 2 in record X:

1. Begin the transaction.
2. FILEC record X with a value of 1.
3. FLFAC FILE=NO for record X.
4. FILEC record X with a value of 2.
5. Commit the transaction.

## Examples

None.

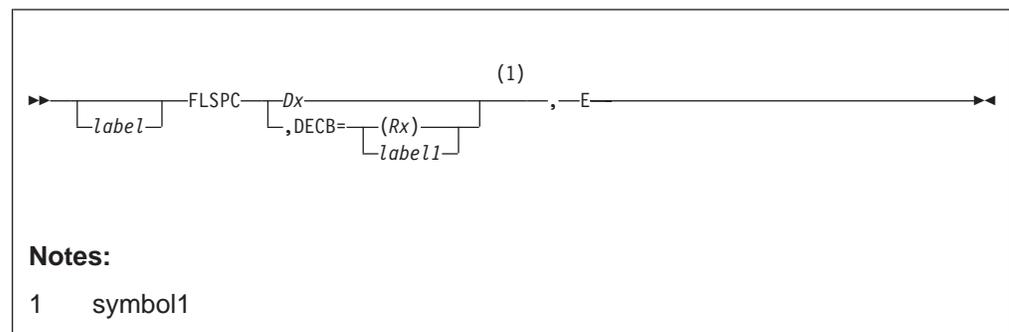
## FLSPC—File a Special Record

Use this system macro to write a single non-TPF-type record that is no longer than 4095 bytes to any cylinder or from main storage to a disk module. The storage block at the specified entry control block (ECB) data level or data event control block (DECB) is not disassociated from the ECB.

The FLSPC macro causes virtual file access (VFA) to be searched for the record. If the record is in VFA, it is forced out of VFA before the new data is written to the DASD.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



*label*

A symbolic name can be assigned to the macro statement.

*symbol1*

Specifies the ECB data level (D0–DF) to be used by the service routine.

**DECB=(Rx)|label1**

The label or general register (R0–R7) containing the address of the DECB, which contains the file address of the record to be filed.

- E** This parameter is required. The file address is in extended mode and contained in the extended file address word defined by symbol1 or the DECB parameter.

## Entry Requirements

- This macro is intended only for control program (CP) support usage.
- R9 must contain the address of the ECB being processed.
- A block of storage must be held by the ECB at the specified core block reference word (CBRW) (CE1CRn) or DECB (IDECDAD).
- The file address must be contained in the specified file address reference word (FARW) (CE1FAn) or DECB (IDECFRW). The record type and record code check (RCC) are optional.
- When accessing a general data set, the appropriate GDSNC or GDSRC macro must be issued to set up the file address before using the FLSPC macro.
- The file address must be in the form BBCCHHR and contained in the extended FARW (CE1FXn) for the ECB data level or DECB (IDECFX0). The bin number

## FLSPC

(BB) should be zeroed and the module number will be the normal halfword number stored in field CE1FMn or field IDECFMOD for the DECB.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- If this macro is called from a program running in 24-bit mode, the condition code will be saved across this macro call. If the macro is called from a program running in 31-bit mode, the condition code upon return from this macro is unknown.
- The FARW at the specified ECB data level (*symbol1*) or DECB is unchanged.
- The status of the operation is unknown.
- The block of storage at the specified ECB data level (*symbol1*) or DECB is not available.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is the only file type macro available for use on cylinder 0, head 0.
- A check is made by the control program (CP) to determine if the ECB is holding a block of storage at the specified ECB data level (*symbol1*) or DECB and if the file address at the specified ECB data level or DECB is valid.
- The operational program may not use the specified level (*symbol1*) or DECB on return from the CP.
- To ensure completion of the operation, a WAITC macro must be issued.
- The number of bytes to be written is located in the byte count field of the CBRW at the specified ECB data level or DECB.
- The core block is available to the operational program upon successful I/O completion.
- If this macro is used on a record defined as a VFA synchronization candidate, the record update will **not** be synchronized with other processors. The record will only be flushed from VFA on the processor from which the macro was issued.
- If the location of the data event control block specified by the DECB parameter does not refer to a valid DECB, control is transferred to the system error routine.

## Examples

The input to write record type SI, with a code check of X'01' and file address of M-20, C- 02, H- 04, R- 09 from data level 1 follows in this example.

```
CE1CR1- 00012480 0031041F      (attached block)
CE1FA1-  E2C901..              (record type, code check)
CE1FM1-  0014....              (module number-mm)
CE1FX1-  00000002 000409..      (bbcchhr.)
```

```
FLSPC      D1,E
```

The input to the SI write record type, with a code check of X'01' and file address of M- 20, C- 02, H- 04, R- 09 from data level 1 follows in this example.

```
CE1CR1- 00012480 0031041F      (attached block)
CE1FA1- E2C901..                (record type, code check)
CE1FM1- 0014....                (module number-mm)
CE1FX1- 00000002 000409..      (bbcchr.)
```

```
LABEL          FLSPC          D1,E
```

The input to the SI write record type with a code check of X'01' and file address of M- 20, C- 02, H- 04, R- 01 from a DECB follows in this example.

```
IDECCT0- 00982000 0031041F      (attached block)
IDECRI0- E2C901..                (record ID, record code check)
IDECFO0- 0014....                (module number-mm)
IDECFO0- 00000002 000401..      (bbcchr.)
```

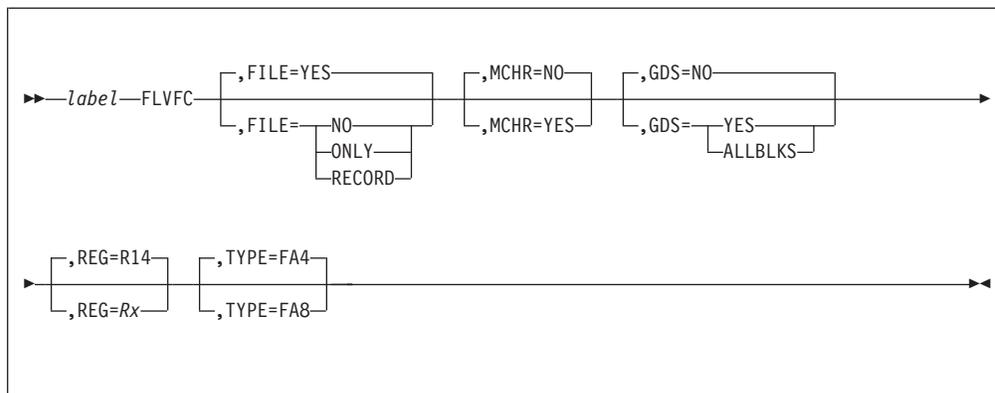
```
FLSPC DECB=(R7),,E
```

## FLVFC—Flush a Record from VFA Buffers

Use this system macro to flush a record from virtual file access (VFA). The next request for this record may force a read from file depending on the options used.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### FILE

Specify one of the following:

##### YES

Requests that the flushed record be filed out to DASD if the record is delayed file pending before it is deleted from VFA. This is a *purge with file down*.

##### NO

Requests the record not be filed out to DASD even if it is delayed file pending. This is a *purge without file down*.

##### ONLY

Requests the flushed record be filed out to DASD if the record is delayed file pending. Otherwise no change is to be made to the record's VFA residence. It will be left as available in VFA. This is only a *file down if delayed filing is pending*.

##### RECORD

Requests a copy of the record, if the record is delay file pending, be passed to the exception recording routines to be exception recorded. The record will remain in VFA in delay file pending status.

If the FILE parameter is omitted, the YES option is assumed.

#### MCHR

Specify one of the following:

##### YES

Requests the flushed record be located using the specified hardware address.

**NO**

Requests the record be located using the supplied FARF address with its associated DBI and SSU values.

When the MCHR parameter is omitted, NO is assumed.

**GDS=NO|YES|ALLBLKS**

This parameter specifies that the file address that is passed is for a general file (GF) or general data set (GDS) record. The ALLBLKS option specifies that all blocks for the data set will be flushed.

When the GDS parameter is omitted, NO is the default.

**REG=R14|Rx**

A general register, one of R0 through R7 or R14 through R15, containing the address of the parameter list defined by the IDSFLV macro.

**TYPE**

Indicates the length of the file address to be flushed. Specify one of the following:

**FA4**

Indicates that the address of the file record to be flushed is a 4-byte file address. The address is located in field IFLVFARF of the parameter list defined by the IDSFLV macro.

**FA8**

Indicates that the address of the file record to be flushed is an 8-byte file address. The address is located in field IFLVFA8 of the parameter list defined by the IDSFLV macro.

## Entry Requirements

- The routine invoking the FLVFC macro must build a parameter list as defined by the IDSFLV macro and pass the parameter list address in the register specified by the REG parameter.
- If MCHR=YES is specified, the TYPE parameter is ignored because the record is specified in the hardware address format and must be present in fields IFLVMOD and IFLVCHR as defined in the IDSFLV macro.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- For a CP invocation, the contents of R0–R13 are preserved across this macro call.
- For a ECB invocation, the contents of R0–R7 are preserved across this macro call.
- The macro service routine will set R14 to contain a hexadecimal return code that can be used by the calling program as follows.
  - R14 = 00 → Record was successfully flushed from VFA
  - R14 = 04 → Record not in VFA
  - R14 = 08 → Record found but was not filed
  - R14 = 0C → Invalid file address
- The contents of R15 are unknown.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called from E-type or C-type programs.

## FLVFC

- If the program to be flushed has been locked in memory (using the GETPC macro), the FLVFC macro will not flush the program until it is unlocked (by using the RELPC macro).
- If a FLVFC macro is issued within a commit scope, the effect of the FLVFC macro is deferred until the root scope commits the transaction.

The processing sequence of file-type macros and FLVFC macros is important. In a commit scope, the FLVFC macro preserves the logical sequence of files and FLVFCs. The following sequence results in a value of 2 in record X:

1. Begin the transaction.
2. FILEC record X with a value of 1.
3. FLVFC FILE=NO for record X.
4. FILEC record X with a value of 2.
5. Commit the transaction.

## Examples

None.

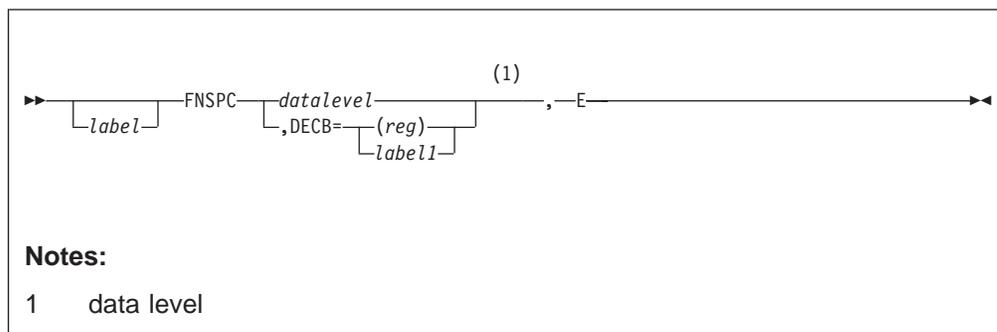
## FNSPC—Find a Special Record

Use this system macro to read a single non-TPF-type record no longer than 4095 bytes from any cylinder. The control program (CP) obtains a 4095-byte block of storage and the address of the block is placed in the core block reference word (CBRW).

The FNSPC macro causes virtual file access (VFA) to be searched for the record. If the record is in VFA, it is forced out of VFA before the record is read from the specified DASD.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *datalevel*

Specifies the entry control block (ECB) data level (D0–DF) that will be used by the service routine.

### **DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB), which contains the file address of the record that will be read.

**E** This parameter is required. The file address must be in extended mode and contained in the extended file address word (FAWX) defined by *datalevel* or DECB.

## Entry Requirements

- This macro is for use in the control program (CP) only.
- R9 must contain the address of the ECB being processed.
- A block of storage must not be held by the ECB at the specified ECB data level (CE1CRn) or DECB (IDECDAD) when the macro is issued.
- The file address must be contained in the specified file address reference word (FARW) (CE1FAn) or DECB (IDECFRW). The record type and record code check (RCC) are optional.
- When accessing a general data set the appropriate GDS macro must be issued to set up the file address before using FNSPC.

## FNSPC

- The file address must be in the form BBCCHHR and contained (left-justified) in the FAWX (CE1FXn) for the ECB data level or DECB (IDECFX0). The bin number (BB) must be zeros. The module number will be the normal halfword number stored in CE1FMn for the ECB data level or IDECFMOD for the DECB.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- If this macro is called from a program running in 24-bit mode, the condition code will be saved across this macro call. If the macro is called from a program running in 31-bit mode, the condition code upon return from this macro is unknown.
- The contents of the CBRW at the specified ECB data level (*datalevel*) or DECB is unknown.
- The FARW at the specified ECB data level or DECB is unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is the only FIND-type macro available for use on cylinder 0, head 0.
- The control program (CP) checks to determine that the ECB is not holding a block of storage at the specified ECB data level or DECB and that the file address specified is valid. If either condition is violated, control is transferred to the system error routine.
- This macro places the core block size of 4095 in the CBRW. The byte count field contains the size of the record actually read. The data obtained is always placed in a 4095-byte core block.
- To ensure that the operation is completed, a WAITC macro must be issued before referencing the data that is read. After WAITC macro processing is completed, the CBRW at the specified ECB data level or DECB contains the address of the record.
- If this macro is used on a record defined as a VFA synchronization candidate, the record update will **not** be synchronized with other processors. The record will only be flushed from VFA on the processor from which the macro was issued.
- If the location of the data event control block specified by the DECB parameter does not refer to a valid DECB, control is transferred to the system error routine.

## Examples

The input to retrieve the SI record type, with a code check of X'01' and file address of M- 20, C- 02, H- 04, R- 01 follows in this example.

```
CE1CR1- ..... ..01....      (no attached block)
CE1FA1- E2C901..           (record type, code check)
CE1FM1- 0014....          (module number-mm)
CE1FX1- 00000002 000401..  (bbcchhr.)
```

```
FNSPC          D1,E
```

The input to retrieve the SI record type with a code check of X'01' and file address of M- 20, C- 02, H- 04, R- 01 for a DECB follows in this example.

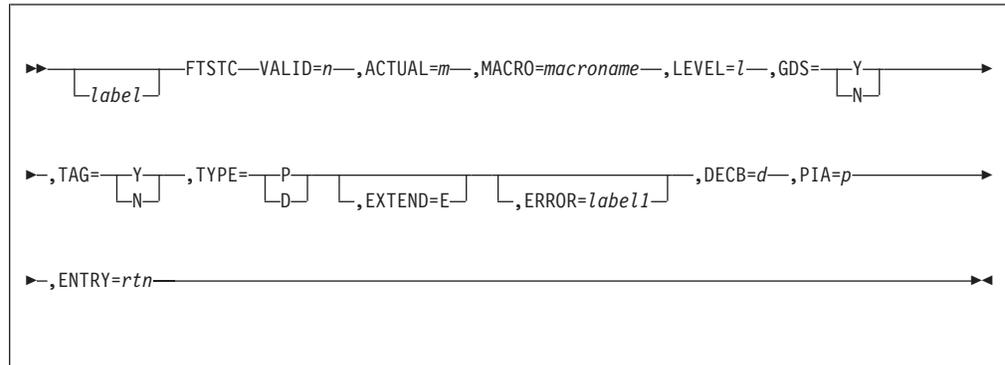
IDECCT0- ..... ..01.... (no attached block)  
IDE1RID- E2C901.. (record ID, record code check)  
IDECFM0- 0014.... (module number-mm)  
IDDECFX0- 00000002 000401.. (bbcchr.)

FNSPC DECB=(R7),,E

## FTSTC–Find/File

Use this system macro to provide parameter validation and transformation, as well as to build the correct linkage for a specific macro. This macro is used by all DASD FIND and FILE-type macros for performing this common processing.

### Format



*label*

The symbolic name from the subject macro statement.

**VALID=*n***

The number of valid positional parameters for the subject macro. This parameter is required.

**ACTUAL=*m***

The number of actual positional parameters entered for the subject macro. The value from the assembler statement N'&SYSLIST, the number of items in the SYSLIST. This parameter is required.

**MACRO=*macroname***

The name of the subject macro (for example the FINDC macro). This parameter is required.

**LEVEL=*l***

The value specified for the data level parameter. D0–DF are the only valid values.

**GDS=Y|N**

The value specified for the GDS parameter, provided the subject macro supports the GDS parameter. Only Y or N are valid values for the GDS parameter.

**TAG=Y|N**

The value specified for the TAG parameter, provided the subject macro supports the TAG parameter. Only Y or N are valid values for the TAG parameter.

**TYPE=P|D**

The value specified for the positional parameter that describes the type of record (primary or duplicate) required for the FINSC and FILSC macros. Only P or D are valid values for the TYPE parameter.

**EXTEND=E**

The value specified for the positional parameter that indicates the use of the extended file address reference word for the FNSPC and FLSPC macros. Only E is a valid value for the EXTEND parameter.

**ERROR=*label1***

The value specified for the error label (*label1*) positional parameter if the subject macro supports the error label positional parameter.

**DECB=*d***

The value specified for the DECB parameter if the subject macro supports the DECB parameter.

**PIA=*p***

The value specified for the PIA parameter if the subject macro supports the PIA parameter.

**ENTRY=*rtn***

The control program (CP) entry point that the macro is supposed to call.

## Entry Requirements

See the applicable macro for more information.

## Return Conditions

See the applicable macro for more information.

## Programming Considerations

- This macro does not validate that the specified parameters are valid for the subject macro. The coder of the subject macro **must not** specify parameters that are not supported by the subject macro. For example, do not specify the TAG parameter for a FINDC macro.
- The FTSTC macro can be used on any I-stream.

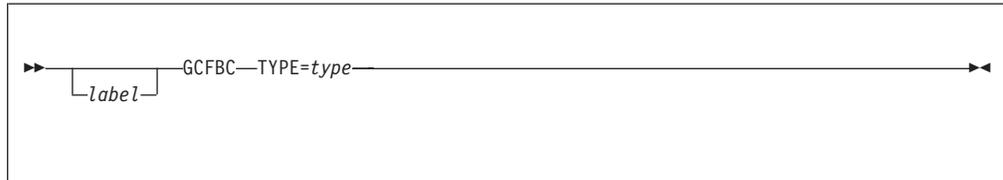
## Examples

None.

## GCFBC—Get Coupling Facility Work Block Address

Use this system macro to get an available coupling facility work block and return the address.

### Format



*label*

specifies a symbolic name that can be assigned to the macro statement.

**TYPE=***type*

specifies the type of coupling facility work block to get, where *type* is CFCB, CFRB, CFSB, or CFVB.

### Entry Requirements

None.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- TPF system processing ends with a catastrophic error if there are no coupling facility work blocks available.
- Register 15 (R15) contains the coupling facility work block address.
- The coupling facility work block type is inserted into the first 4 bytes of the obtained block. The remaining bytes in the block are set to zeros.

### Programming Considerations

- This is a restricted use macro.
- This macro can be processed on any I-stream.
- Enter the RCFBC macro to release the coupling facility work block when you are done using it.

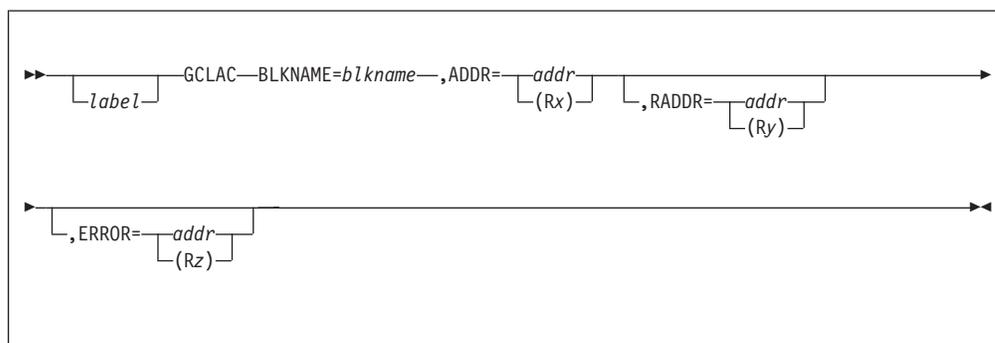
### Examples

None.

## GCLAC—Get a Specified CLAW Block Type

Use this system macro inline to establish addressability to the Common Link Access to Workstation (CLAW) control blocks.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLKNAME=blkname**

The name of the block being requested. BLKNAME is a required parameter.

#### **FOURKF**

Locked page for channel control word (CCW)

#### **ICADAP**

Adapter control block

#### **ICLAWB**

CLAW device interface block

#### **ICLAWG**

CLAW page structure

#### **ICLCON**

Connection control table

#### **ICLIBK**

Client control block

#### **ICLIOI**

Extension block for CLAW I/O interrupt

#### **ICLTRB**

Transaction control block

#### **ICMSGB**

Message control block

#### **ICNBLK**

Extension block for CLAW initialization

#### **ICPATH**

Path control block

#### **ICPERM**

Permanent work area

#### **ICPOLL**

POLL request extension block

## GCLAC

### **ICQBLK**

Message queue element structure

### **ICRBLK**

Extension block for returning CLAW page

### **ICRCCW**

Read channel CCW area

### **ICTRCE**

Trace data structure

### **ICWCCW**

Write channel CCW area

### **ISCCDT**

CLAW device table

### **ISCFDT**

File descriptor

### **ISCIPT**

Internet Protocol (IP) address table.

### **ADDR**

The SVM address of the requested block. The ADDR parameter is required.

*addr*

An address.

### **(R<sub>x</sub>)**

A register containing the address.

### **RADDR**

The system virtual memory (SVM) address of the requested block. The RADDR parameter is optional.

*addr*

The SVM address.

### **(R<sub>y</sub>)**

A register containing the address.

### **ERROR**

If an error occurs, processing resumes at the specified address. The ERROR parameter is optional.

*addr*

An address.

### **(R<sub>z</sub>)**

The register containing the address of the error.

## Entry Requirements

This macro is for use in the control program (CP) only.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- If an error occurs (for example, if no blocks of the specified type are available), control resumes with the address specified by the ERROR parameter or the program exits.

- The contents of all registers, except R0, R14, and R15, are preserved across this macro call.
- R0 contains the return code. A zero return code indicates success and a nonzero return code indicates an error.

## Programming Considerations

This macro can be run on any I-stream.

## Examples

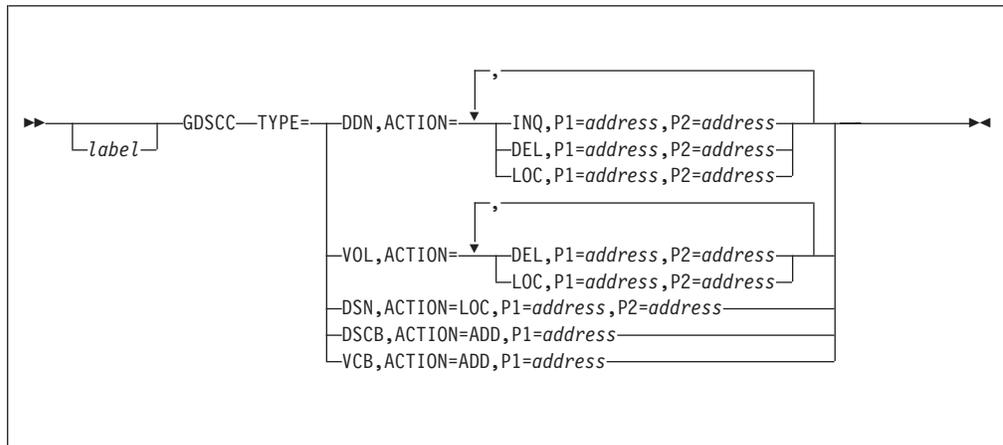
- This call shows block ICLAWG being requested for address VCTADDR.  
`GCLAC BLKNAME=ICLAWG,ADDR=VCTADDR`
- This call shows block ICLAWG being requested for the address found in register R3.  
`GCLAC BLKNAME=ICLAWG,ADDR=(R3)`

## GDSCC—General Data Set (GDS) Control

Use this system macro to to:

- Maintain a control structure for general data sets (GDSs)
- Locate, add, and delete data sets, volumes, and data definition names from the GDS storage structure
- Inquire about data sets, volumes, and data definition names.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### TYPE

The object that is being acted upon. Valid values are:

##### DDN

A 16-character data definition name (DDNAME)

##### VOL

A volume number

##### DSN

A data set name (DSNAME)

##### DSCB

A data set control block (IDSDSB)

##### VCB

A volume control block (IDSDSB).

#### ACTION

The action to be performed. Valid values are:

##### INQ

Retrieve information about a specific DDNAME from the GDS control structure. The valid type is DDN.

This value of the ACTION parameter is available for general use.

##### P1=address

The address of the DDNAME for which information is to be retrieved. The DDNAME field must be 16 bytes in length and be padded on the right with blanks.

**P2=address**

The address of a control block, as defined in the `IDSINQ` macro, that is to be used by the TPF system to return information about the specified DDNAME.

**ADD**

Add the specified control block to the GDS structure. Valid type codes are DSCB or VCB.

This value of the ACTION parameter is restricted to system only use.

**P1=address**

The address of the DSCB/VCB (IDSDSB) block to be added.

**P2=address**

Not used

**DEL**

Delete specified control information from the GDS control structure. The valid types are DDN or VOL.

This value of the ACTION parameter is restricted to system only use.

**P1=address**

The address of the DDNAME to be acted on for types DDN and VOL. The DDNAME field must be 16 bytes in length and be padded on the right with blanks.

**P2=address**

The address of a fullword volume number is required for type VOL.

**LOC**

Locate specified control information for the specified input. Valid types are DDN, VOL, or DSN.

This value of the ACTION parameter is restricted to system only use.

**P1=address**

The address of the DDNAME or DSNAME to be acted on for types DDN, DSN, and VOL. The DDNAME field must be 16 bytes in length and be padded on the right with blanks. The DSNAME field must be 44 bytes in length and be padded on the right with blanks.

**P2=address**

The address of a fullword volume number for type VOL.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- R14 contents is dependent on the action code.

Action Code	Description
ADD	The address of the system block for the DSCB for type DDN, DSN, or DSCB.
	The address of the system block for the VCB for type VOL or VCB.
DEL	The remaining count of mounted data set volumes.

## GDSCC

INQ	Unpredictable.
LOC	The address of the system block for the DSCB for type DDN or DSN. The address of the system block for the VCB for TYPE or VOL.

- R15 contains the return code.  
Information about the ADD, DEL, and LOC action codes follows.

Action Code	Description
0	If successful.
4	If the name was not found.
8	If the volume was not found.
12	If there was a logic error.

Information about the INQ action code follows.

Action Code	Description
0	If successful.
4	If the name was not found.

## Programming Considerations

When the ADD, DEL, or LOC actions are performed, this macro can be run only on the main I-stream. When the INQ action is performed, this macro can be run on any I-stream.

## Examples

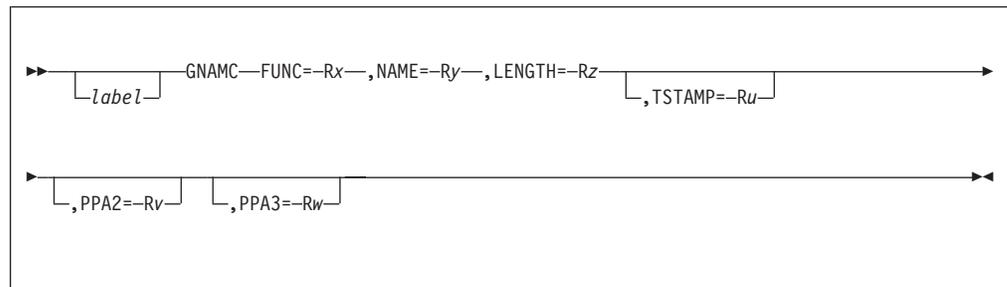
None.

## GNAMC–Get Program Prolog Area (PPA) Functional Name Information

Use this system macro to find information about the following from the program prolog area-1 (PPA1):

- Address of the PPA1
- Address of the PPA1 field that contains the function name
- Length of the function name
- Address of the timestamp block
- Address of the PPA2
- Address of the PPA3.

### Format



#### *label*

A symbolic name assigned to the macro statement.

#### **FUNC=R $x$**

This input parameter register, other than register 0, contains the address of an ISO-C function.

On output, this register contains one of the following:

- The address of the PPA1
- Zeroes when the contents of the PPA1 are not valid.

**Note:** The address of a valid PPA1 is returned for both the Language Environment and for the non-Language Environment. For the Language Environment, the Language Environment routine layout entry is used to find the address of the valid Language Environment PPA1.

#### **NAME=R $y$**

This output parameter register, other than register 0, contains one of the following:

- The address of the first character of the function name
- Zeroes when the function name does not exist or when the contents of the PPA1 are not valid.

#### **LENGTH=R $z$**

This output parameter register, other than register 0, contains one of the following:

- The length of the function name
- Zeroes when the function name does not exist or when the contents of the PPA1 are not valid.

## GNAMC

### TSTAMP=R<sub>u</sub>

This optional output parameter register, other than register 0, contains one of the following:

- The address of the timestamp block
- Zeroes when the timestamp block is not available or when the contents of the PPA1 or the PPA2 are not valid.

### PPA2=R<sub>v</sub>

This optional output parameter register, other than register 0, contains one of the following:

- The address of the PPA2 (Compile Unit Block)
- Zeroes when the PPA2 is not available or the PPA1 is not valid.

### PPA3=R<sub>v</sub>

This optional output parameter register, other than register 0, contains one of the following:

- The address of the PPA3 (Debug Unit Block)
- Zeroes when the PPA3 is not available or the PPA1 is not valid.

## Entry Requirements

The GNAMC macro does not support the TARGET(TPF) compiler.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- All registers are preserved across this macro call with the exception of the registers specified on the parameters.

## Programming Considerations

- The FUNC parameter register must point to a valid ISO-C function, for example:
  - A valid Language Environment routine layout entry
  - A valid PPA1 area in a non-Language Environment.
- If a valid PPA1 is not found, zeroes are returned in the:
  - FUNC parameter register
  - NAME parameter register
  - LENGTH parameter register
  - TSTAMP parameter register if specified
  - PPA2 parameter register if specified
  - PPA3 parameter register if specified.
- If a function name does not exist in the PPA1, zeroes are returned in the:
  - NAME parameter register
  - LENGTH parameter register.

## Examples

- In this example, register 15 contains the address of an ISO-C function. This address is the address of a Language Environment routine entry layout.

```
GNAMC FUNC=R15,NAME=R4,LENGTH=R6
```

This invocation returns the following:

- The address of the PPA1 in register 15.

- The address of the first character of the function name in register 4.
- The length of the function name in register 6.
- In this example, register 15 contains the address of an ISO-C function. This address is the address of a non-Language Environment PPA1 area.

GNAMC FUNC=R15,NAME=R2,LENGTH=R7,TSTAMP=R4,PPA2=R5,PPA3=R6

This invocation returns the following:

- The address of the PPA1 in register 15.
- The address of the function name in register 2.
- The length of the function name in register 7.
- The address of the timestamp area in register 4.
- The address of the PPA2 in register 5.
- The address of the PPA3 in register 6.

### GROUP–Recoup Descriptor Record Access

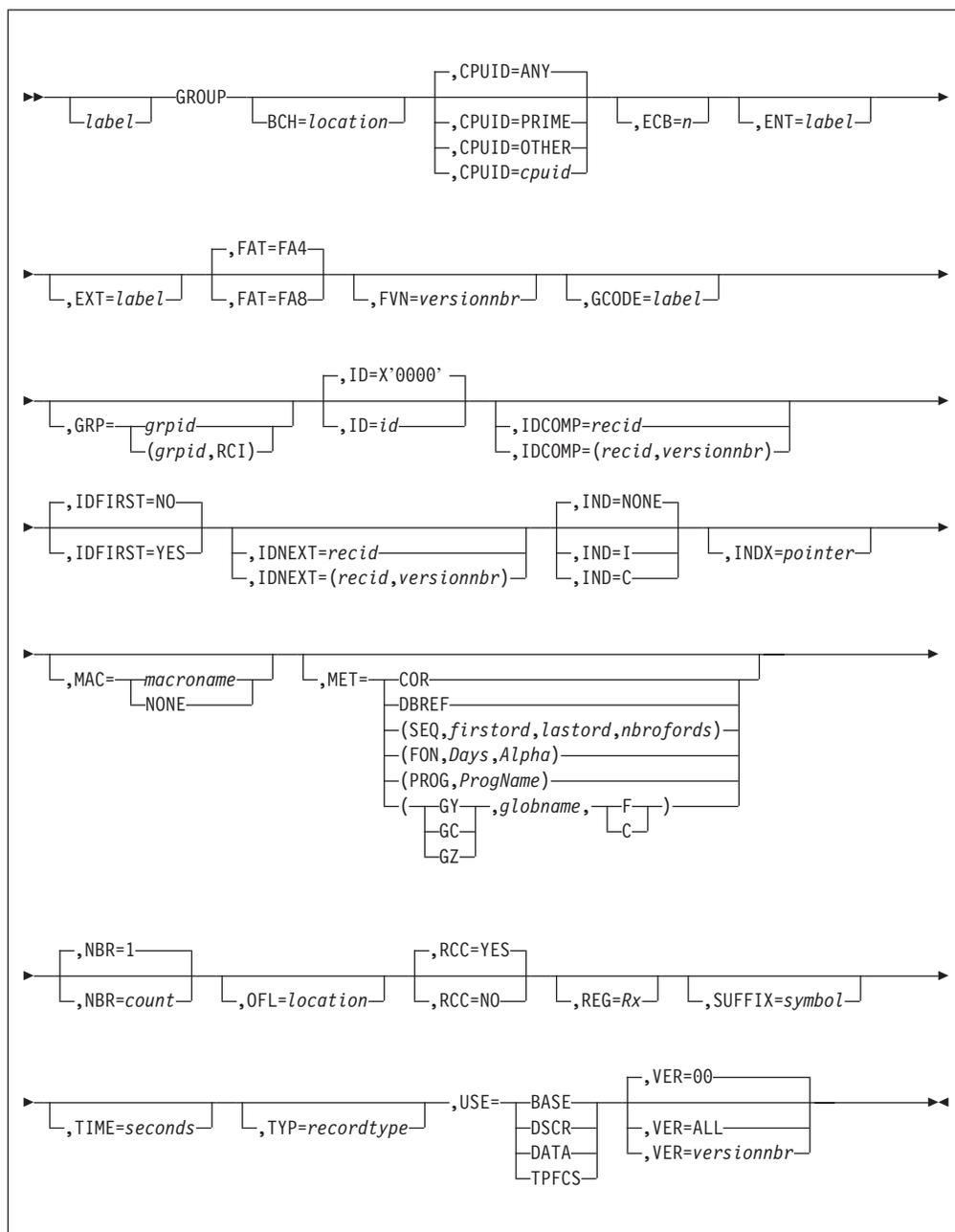
Use this system macro to do the following:

- Calculate fixed file addresses
- Find main storage records
- Locate records using globals.

Together, the GROUP and INDEX macros are used offline to build descriptor (BKD) records, which can be loaded to the online TPF system. These descriptors provide instructions, values, and displacements that are used by recoup processing to chain chase data structures and check which file pools are being used. See *TPF Database Reference* for more information about descriptors.

**Note:** Be careful when you use the GROUP and INDEX macros because these macros are required for recoup processing. Coding the GROUP or INDEX macro incorrectly can result in the loss of pool records that contain valid data.

## Format



*label*

specifies a symbolic name that can be assigned to the macro statement.

**BCH=location**

specifies the location of the backward chain, where *location* is a field name or an absolute displacement as shown in the following examples:

```
BCH=TAGLOC
BCH=(N,32)
```

BCH is specified only when the backward chain is the last-in chain and when the OFL parameter is coded.

## GROUP

### CPUID

specifies the ID of the processor where chain chase processing is to be performed. If the specified CPU is not active, the primary processor performs the chain chase at the end of recoup phase 1.

The CPUID parameter can specify one of the following:

#### ANY

specifies that chain chase processing for this record ID occurs on any processor that is used in the recoup run.

#### PRIME

specifies that chain chase processing for this record ID occurs on the primary recoup processor.

#### OTHER

specifies that chain chase processing for this record ID occurs on any processor other than the primary processor that is used in the recoup run.

#### *cpuid*

is the specific processor where chain chase processing for this record ID occurs.

### ECB=*n*

specifies the maximum number of entry control blocks (ECB) that will be active to chain chase the primary (or anchor) records for this descriptor number, where *n* is a decimal number.

The actual number of ECBs that may be active for this descriptor may be greater depending on how the descriptor macros are coded.

**Note:** The number assigned to this parameter must be less than the maximum number of ECBs that recoup will allow to be active (values BZNESR and BZNEST in the BRPEQ macro)

If USE=BASE is specified and no ECB parameter is specified, *n* defaults to BZNESR-1.

If USE=DSCR is specified and no ECB parameter is specified, *n* defaults to 0.

### ENT=*label*

specifies exit code to run before processing the record type, where *label* is the label of the start of the exit code. You can use user exit code to do the following:

- To set a global variable indicating that processing has started for a particular record type
- To set a global variable indicating to the queue package that get file storage (GFS) will be used when moving a message from one queue to another
- To set the recoup keypoint (BK0EEX) bit to zero to prevent chain chasing a descriptor
- To do anything else before processing the record.

**Note:** You can only use the ENT parameter with the USE=BASE parameter.

### EXT=*label*

specifies code to run when record type processing is completed, where *label* is the label of the start of the code. One use of the EXT parameter is to run code to reverse an indicator set by the code called with the ENT parameter.

**Note:** You can only use the EXT parameter with the USE=BASE parameter.

**FAT**

specifies the format of the file address located by the OFL and BCH parameters. The FAT parameter can be used only when the OFL parameter is coded. The following options can be coded:

**FA4**

specifies a 4-byte file address.

**FA8**

specifies an 8-byte file address.

**FVN=versionnbr**

specifies the file version number used by the TPFDF product to identify file structures that have blocks with different layouts than the prime block, where *versionnbr* is a number from 0 to 254. You can only specify the FVN parameter when MET=DBREF is specified.

**GCODE=label**

specifies code that will be run for fixed file records only, where *label* is the label of the start of the code.

Example: A fixed record contains some data indicating that there is no active structure (of pool records) currently attached to the fixed record. You can bypass chain chase processing from this particular fixed record for this recoup run.

**GRP**

specifies that this and all other descriptors with the same group ID are chain chased on the same processor.

*grpid*

is the 2-character EBCDIC or 4-character hexadecimal group ID that is assigned to a specific group of record IDs.

**RCI**

specifies that the related group is chain chased using recoup chain chasing indicator (RCI) processing. RCI processing uses a special set of pseudo directories and a different chain chase algorithm.

**Notes:**

1. You can only use the RCI parameter with the USE=BASE parameter.
2. Specify the DUPEELIM=YES parameter on the associated INDEX macro.

**ID=id**

specifies the ID of the fixed record, where *id* is a 2-character EBCDIC or 4-character hexadecimal record ID.

**IDCOMP**

specifies the ID of the record for which chain chasing must be completed before chain chasing this descriptor.

*recid*

is a 2-character EBCDIC or 4-character hexadecimal record ID.

*versionnbr*

is a version number, from 0 to 254, used to identify file structures that have different layouts.

**IDFIRST**

specifies one of the following:

## GROUP

### NO

specifies that this record will be chain chased after all records with IDFIRST=YES specified.

### YES

specifies that this record and any other records with IDFIRST=YES specified must be chain chased before other records are chased.

### IDNEXT

specifies the ID of the record for which chain chasing will start when chain chasing is completed for this record.

### IND

specifies the following indicators:

#### NONE

specifies that there are no indicators.

**I** instructs the chain chase scheduler to ignore this GROUP macro and all statements associated with it.

**C** specifies that the recoup descriptor belongs to TPF collection support (TPFCS) and only TPFCS will process this descriptor.

### INDX=*pointer*

specifies a pointer to a common INDEX statement. This applies only when there is one INDEX statement per GROUP macro.

The INDX parameter can only be used when USE=BASE is coded on the GROUP macro.

When the DSCR parameter in the INDEX macro points to a GROUP macro, it indicates that a pool record has embedded pool addresses and needs additional description by a new set of descriptor parameters. When this occurs, the only parameters necessary on the GROUP macro are MAC, ID, and NBR.

### MAC

specifies one of the following:

#### *macroname*

is the name of the data macro describing the data record containing embedded addresses. The MAC parameter is required when USE=BASE is specified and can be used when USE=DATA is specified.

#### NONE

specifies that no data macro describes the data record containing embedded addresses.

### MET

specifies the method used for chain chasing. You must specify the MET parameter when USE=BASE is specified, but it is not necessary when USE=DSCR is specified.

### COR

specifies that the fixed record is located in main storage and loaded by the primary loading segment for globals (GOGO) using the global allocator record (GO1GO). The correct global ordinal number for the global storage allocator record (GOA) must be in BAM1. No additional information is needed when MET=COR.

### DBREF

specifies that the record is a TPFDF record to be chain chased by TPFDF recoup.

**Notes:**

1. You can only specify MET=DBREF when USE=DSCR is specified.
2. TPFDF recoup runs under the control of TPF recoup.
3. You may need TPFDF DBDEF user exit code to prevent the TPFDF product from chain chasing this structure more than once.

**SEQ**

specifies that the fixed records are sequential and are incremented by 1 to obtain the next ordinal number.

*firstord*

is the first ordinal number. The default is 0.

*lastord*

is the last ordinal number. If the *lastord* parameter value is not specified, processing continues until a file address compute (FACE) program error is indicated.

*nbrofords*

is the total number of ordinals to process. If the *nbrofords* value is not specified, processing continues until a FACE program error is indicated.

**(FON,Days,Alpha)**

specifies the flight ordinal number (FON), where *Days* is the number of days in the current or gross period and *Alpha* is the number of alpha groups. The following table shows an example of variable values that could be used for record types that are unique to a programmed airline reservation system (PARS) application.

<b>Record Type</b>	<i>Days</i>	<i>Alpha</i>
PNID and ING	1	Total number of ING records assigned for each flight.
IND	1	Number of days in the detail period.
INM	1	1
PNIG	1	Number of Alpha groups in the TPF system (#ALPHA)
TSS	1	1

**(PROG,ProgName)**

specifies the name of the program that will be entered to process the fixed record. This will be a special user-written program. The name must be 4 characters long.

**GZ**

specifies that the address of the record is in global 1.

**GY**

specifies that the address of the record is in global 3.

**GC**

specifies that the address of the record is in the common global (P or Q).

*globname*

is the global name that has the address of the record; for example, @NSCKAD.

**F** specifies that the address is a file address.

**C** specifies that the address is a main storage (core) address.

## GROUP

### **NBR**=*count*

specifies the number of INDEX macros to follow the GROUP macro, or the number of different record structures whose addresses are chained from or embedded into the fixed record, where *count* is a decimal number.

**Note:** Do not include in the count, INDEX macro statements that specify the ALTID parameter.

### **OFL**=*location*

specifies the location that contains the file address of the overflow of the fixed record, where *location* is a field name or an absolute displacement as shown in the following examples:

```
OFL=TAGLOC  
OFL=(N,24)
```

**Note:** If the overflow is not identical in format to the prime or fixed record, describe the overflow with an INDEX macro.

### **RCC**

specifies the record code check indicator (RCC). If the value specified with the RCC parameter is not YES or NO, YES is assumed. If the RCC parameter is not specified, NO is assumed.

#### **YES**

specifies that the RCC of the prime record is copied to the file address reference word (FARW) before searching for the overflow records in segments BRFM and BPM0.

#### **NO**

specifies that the RCC of the prime record is not copied to the FARW in the same segments.

### **REG**=*Rx*

specifies the base register used during online processing when USE=DATA is specified.

### **SUFFIX**=*symbol*

specifies a symbol to use during online processing to tell the difference between DSECTS. The default is a blank character.

### **TIME**=*seconds*

specifies the amount of time to allow all the ECBs to complete chain chasing this structure before starting timeout processing, where *seconds* is a number of seconds from 0 to 32 767. This parameter is required when USE=BASE is specified and defaults to 0 when USE=DSCR is specified.

**Note:** Timeout processing will not occur if you are running recoup in 1052 state. Operator intervention is required to stop processing a descriptor that takes too long to be completed.

### **TYP**=*recordtype*

specifies the symbolic FACE record type (for example, #WAARI) of the fixed record.

#### **Notes:**

1. You must specify the TYP parameter when MET=SEQ or MET=FON is specified.
2. If you specify USE=BASE without specifying the TYP parameter, the record type defaults to 0.

**USE**

specifies one of the following:

**BASE**

specifies that the GROUP macro describes a record that is a fixed record or the anchor record of the data structure being chain chased.

**DSCR**

specifies that the GROUP macro describes a record that was retrieved as an embedded pool address of another record that was defined previously.

**DATA**

specifies that the GROUP macro is used online and is generating the group macro data DSECT only.

**Note:** USE=DATA is used only for online processing.

**TPFCS**

specifies that the GROUP macro is used for TPFCS recoup only.

**Note:** TPF recoup will ignore these group descriptors while doing chain chasing. You must ensure that all IDs used with USE=TPFCS are unique.

**VER**

specifies the ID version number of a TPF file; used for chain chasing records with the same ID but different data. You cannot specify the VER parameter if MET=DBREF is specified.

**00** specifies that version 0 of the file is chain chased.

**ALL**

specifies that all versions of the file are chain chased.

*versionnbr*

is a version number from 0 to 254 of the file that is chain chased.

## Entry Requirements

Because this macro is declarative, there are no entry requirements.

## Return Conditions

None.

## Programming Considerations

- This macro can be run from any I-stream.
- The GROUP macro can be used only by recoup segments.
- Observe the following for offline programs:
  - This macro is used to create descriptor segments with the USE=BASE or USE=DSCR parameters. These segments are loaded to the TPF system by using the ZOLDR LOAD command.
  - All parameters appropriate to the record being defined by this GROUP macro must be included in the macro statement.
  - Ensure that the DSECT referred to by the MAC parameter describes the subject record and uses register R0 as its base register.
- For online programs:
  - This macro is used to describe the different fields in the BKD record by using the USE=DATA and REG=Rx parameters.

## GROUP

- The BKD records created by GROUP and INDEX macros are moved into the BKD load control record (BK0LC) by the ZRBKD command and read by the remaining recoup segments.
- Register 9 (R9) must contain the address of the ECB being processed.

## Examples

- The following example shows online use of the GROUP macro:

```
GROUP REG=R7,USE=DATA
```

REG=R7 Base register for the online use.  
USE=DATA Indication that this GROUP call is for online use.

- The following examples show offline use of the GROUP macro.

The following example shows the GROUP macro describing a fixed or anchor record of the data structure being chain chased:

```
GROUP MAC=LK4KC,ID=00FF,ECB=10,MET=(GY,@NSCKAA,F),  
TIME=420,NBR=4,GCODE=LINKCHK,USE=BASE,VER=11
```

MAC=LK4KC LK4KC is the name of data macro describing the record.  
ID=00FF 00FF is the record identification of the fixed record.  
ECB=10 No more than 10 ECBs to chase the anchor records  
MET=(GY,@NSCKAA,F) GY is the address of a global record and  
@NSCKAA is the global name that has the address.  
F indicates the address is a file address.  
TIME=420 420 seconds will be allowed for all ECBs to complete  
processing of this structure.  
NBR=4 4 INDEX statement will follow this GROUP statement.  
GCODE=LINKCHK LINKCHK is the label of user exit code.  
USE=BASE This GROUP macro is describing a fixed record.  
VER=11 This is version 11 of this record.

The following example shows the GROUP macro describing a record that was an embedded pool of another record defined previously:

```
GROUP MAC=MC0MC,ID=MC,NBR=1,USE=DSCR
```

MAC=MC0MC MC0MC is the name of data macro describing the record.  
ID=MC MC is the record identification of the pool record.  
NBR=1 1 INDEX statement will follow this GROUP statement.  
USE=DSCR This GROUP macro is describing a record that was  
retrieved as an embedded pool address of another record.

The following example shows the GROUP macro describing a record that was an embedded pool of another record defined previously but defaults are used:

```
GROUP ID=OM,USE=DSCR
```

ID=OM OM is the record identification of the pool record.  
USE=DSCR This GROUP macro is describing a record that was  
retrieved as an embedded pool address of another record.  
Defaults were used were ECB=0, MAC=Same data macro of previous  
record, NBR=1, SUFFIX=' ', TIME=0.

The following example shows the GROUP macro describing a fixed or anchor record sequential in nature.

```
GROUP MAC=CI0CO,ID=CO,ECB=10,MET=SEQ,TYP=#RCBRA,  
NBR=4,TIME=480,USE=BASE
```

MAC=CI0CO CI0CO is the name of data macro describing the record.  
ID=CO CO is the record identification of the fixed record.  
ECB=10 No more than 10 ECBs to chase the anchor records  
MET=SEQ The method of access is sequential and defaults are used  
for first, last and ord subparameters. Starts with  
ordinal 0 and ends with a FACE error. Processing

```

                will continue until a FACE error is indicated.
TYP=#RCBRA   #RCBRA is the FACE record type.
TIME=480     480 seconds will be allowed for all ECBs to complete
              processing of this structure.
NBR=4        4 INDEX statement will follow this GROUP statement.
USE=BASE     This GROUP macro is describing a fixed record.

```

- The following examples show the GROUP and INDEX macros being used together.

The following example shows how the GROUP and INDEX macros are used together online:

```

GROUP REG=R7,USE=DATA
INDEX REG=R2

```

This example shows the use of the GROUP and INDEX to access different fields in the BKDx records.

- The following examples show how the GROUP and INDEX macros are used offline to build BKDx records. The emphasis of these examples is on the relationship between the GROUP and INDEX macros to logically describe a structure.

GROUP and INDEX macros are combined to describe fixed and pool record structures currently existing on a user database. Together, they logically describe a structure.

```

AAA  GROUP USE=BASE,NBR=2
      INDEX TYP=C
      INDEX TYP=F,DSCR=BBB
BBB  GROUP USE=DSCR,NBR=2
      INDEX TYP=C
      INDEX TYP=F,DSCR=CCC
CCC  GROUP USE=DSCR,NBR=1
      INDEX TYP=C

```

In offline programs, GROUP and INDEX macros are used to build BKDx records.

The USE parameter on the GROUP macro establishes a particular data record as either fixed (USE=BASE) or pool (USE=DSCR). Subsequent INDEX macros related to the GROUP macro immediately preceding it describe additional embedded pools that are chained together by coding its DSCR parameter. This DSCR parameter points to a subsequent GROUP macro with its USE parameter coded as USE=DSCR. The subsequent GROUP macro would then have related INDEX macros that would either continue the chained data record structure by using its DSCR parameter coded to point to a subsequent group (for example, AAA to BBB to CCC), or would end the chain of data records by eliminating the DSCR parameter from the subsequent INDEX macro at any point in a data record chain.

The following example has two anchor chains, XRRT and NCB, indicated by the USE=BASE parameter.

```

                BEGIN NAME=xxxx,VERSION=zz,IBM=YES
DC  AL2(2)          NUMBER OF DESCRIPTORS
DC  S(XRRT)         FIRST DESCRIPTOR
DC  S(NCB)          SECOND DESCRIPTOR
XRRT GROUP MAC=UT2RT,ID=UT,ECB=31,TYP=#UT2RT,MET=SEQ,
      NBR=1,TIME=240,USE=BASE
      INDEX TYP=V,FI=UT2DAT,FA=(,UT2RRS-UT2DAT),CNT=(2,UT2EID),
      ID=US,DSCR=XRRS,LI=L'UT2DAT
XRRS GROUP ID=US,NBR=1,USE=DSCR
      INDEX TYP=C,ID=US
NCB  GROUP MAC=NC0CB,USE=BASE,ID=CB,ECB=32,MET=SEQ,
      TYP=#NCBRI,NBR=2,TIME=600

```

## GROUP

```
INDEX TYP=F,FA=NC0FRST,ID=MI
INDEX TYP=F,FA=NC0CUR,ID=OM,DSCR=OM
OM   GROUP USE=DSCR,NBR=1,ID=OM
INDEX TYP=C,ID=OM
FINIS
END
```

- The MAC parameter indicates the DSECT that describes this record and the TYP parameter describes the FACE record type.
  - The NBR=1 parameter on the XRRT GROUP and NBR=2 on the NCB GROUP indicate the number of INDEX statements associated with this GROUP macro.
  - Anchor XRRT has one INDEX statement, which is TYP=V. Type V describes a group of embedded addresses and can be one of three variations defined by other parameters on the INDEX statement.
  - The FA parameter describes the displacement to the address in the anchor.
  - The FI parameter describes the displacement to the first item. This INDEX describes an address embedded in a record in the anchor.
  - The DSCR=XRRS parameter on this INDEX statement indicates that a GROUP macro with the name XRRS will define the structure of the pool record and GROUP XRRS has parameter USE=DSCR referring to this.
  - The XRRS GROUP does not have a MAC parameter, so it will be described by the same macro that was used by the previous GROUP macro. The NBR=1 parameter for this GROUP macro indicates it will have an INDEX statement following. The INDEX statement is TYP=C. TYPE C describes the embedded pool address contained at location 8.
  - Anchor NCB has two INDEX statements which are both TYP=F. Type F indicates that the file address is at a certain displacement in the fixed record. In this example, the two locations are NC0FRST and NC0CUR. Parameter DSCR=OM on the INDEX statement associated with location NC0CUR indicates that the GROUP with the name OM will define the structure of the pool record and GROUP OM has parameter USE=DSCR referring to this. The OM GROUP does not have a MAC parameter, so it will be described by the same DSECT that was used by the previous GROUP.
  - The NBR=1 parameter for this GROUP indicates that it will have an INDEX statement following. This INDEX statement is TYP=C, which describes the embedded pool address contained at location 8 in the record.
- The following example shows the GROUP macro with the USE=TPFCS and IND=C parameters specified, which indicates that the GROUP macro is used for TPFCS recoup only:

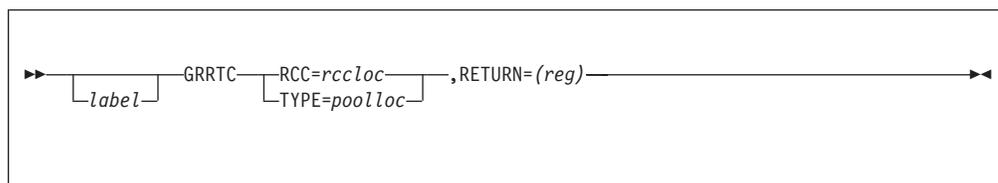
```
TPFCS5  GROUP ID=FC13,NBR=1,IND=C,USE=TPFCS
INDEX TYP=F,FA=( ,8),ID=FC13
```

## GRRTC–Get Record Code Check (RCC) Reference Table

Use this system macro to obtain attributes of pool types and pool sections. This macro returns a pointer to an item in the record code check reference table (IRCCR). Attributes of the item include the following:

- Pool section name
- Pool section device type name
- Whether the pool section is active
- Whether the pool section is long-term or short-term
- Whether the pool section is duplicated records.

### Format



**RCC=rccloc**

specifies the location of the 1-byte RCC.

**TYPE=poolloc**

specifies the location of the 3-byte pool type name.

**RETURN=(reg)**

specifies the register that will contain the address of the IRCCR item whose address is to be returned. The address returned in this register points to a data area structure according to DSECT IRCCR. The specified register cannot be used as a base register for the TYPE or RCC parameters.

### Entry Requirements

This macro will not run when the caller is in 24-bit mode.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- If the requested pool section or pool type does not exist, the register coded in parameter RETURN=(reg) will contain 0.
- The contents of register 14 (R14) and register 15 (R15) are unknown unless specified as the return register
- On return, the register specified in the RETURN parameter will be mapped to the IRCCR DSECT.

### Programming Considerations

This macro can be run on any I-stream.

### Examples

None.

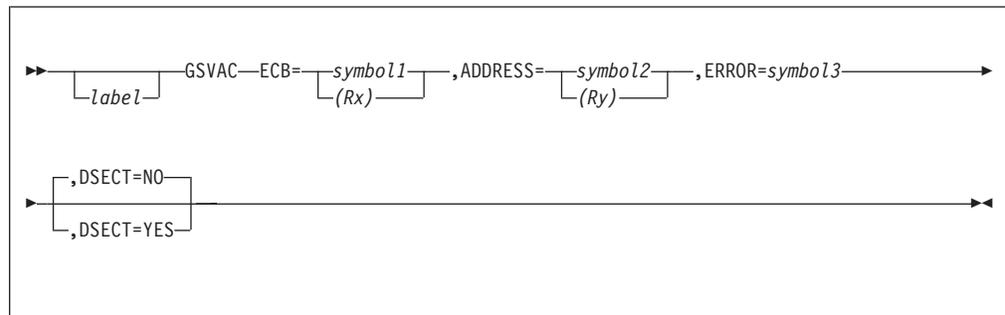
## GSVAC—Convert an EVM Address to an SVM Address

Use this system macro to convert a specified entry control block (ECB) virtual memory (EVM) address (EVA) in a specified EVM to the corresponding system virtual memory (SVM) address (SVA).

An ECB-controlled program can issue this macro to convert any valid EVA in any active EVM on any active I-stream, and the SVA returned is valid for the SVM of the I-stream on which the issuing program is active.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ECB**

This required parameter designates the target EVM and determines the segment and page tables that will be used for conversion. The SVM address of the target ECB may be passed either in a register or in an addressable 4-byte field. The parameter is not modified.

#### *symbol1*

If specified, it is the label of the field that contains the SVM address of the target ECB.

#### **(Rx)**

If specified, it is the general register (R0 through R7) that contains the SVM address of the target ECB.

#### **ADDRESS**

This required parameter specifies the EVM address to be converted. This address may be passed either in a register or in an addressable 4-byte field. The converted address will replace the input parameter.

#### *symbol2*

If specified, it is the label of the field that contains the EVM address to be converted.

#### **(Ry)**

If specified, it is the general register (R0 through R7) that contains the EVM address to be converted. It can not be the same register as coded for Rx.

**ERROR=***symbol3*

This required parameter specifies a program label, in the issuing program, where control is transferred when the designated address cannot be converted. Register R15 will contain an error code indicating the type of error.

**DSECT=NO|YES**

The DSECT parameter is used to generate the DSECT that describes the macro expansion, and certain equates associated with the macro. It is intended for CP use only. The default is NO.

## Entry Requirements

- The ECB-controlled program must be authorized to issue restricted use macros.
- The TPF system should be paused during processing of this macro in order to assure the validity of the converted address. The pause should be ended as soon as possible following processing of the macro.  
If an ECB-controlled program issues the GSVAC macro to convert an EVA in its own EVM, the TPF system need not be paused.
- The target EVM can be processing on any I-stream.

## Return Conditions

- When conversion is successful, control is returned to the next sequential instruction (NSI). The converted address is returned in the field or register designated by the ADDRESS parameter.
- When conversion is unsuccessful, processing continues with the routine specified by the ERROR parameter and error conditions are posted in the low-order byte of general register 15. The specific error can be determined by storing the error byte and testing it with the equate symbols that follow. The following equates are generated with the first use of the macro:

GSVA_ECB_INACT	Target ECB is inactive
GSVA_INVALID_ECB_ADDR	ECB address is not valid.
GSVA_INVALID_EVM_ADDR	EVM address is not valid.

- The contents of registers R14 and R15 are unknown, except on an error condition as described previously.

## Programming Considerations

- In general, the SVM address returned by this macro may not be directly usable by the requesting program. It is recommended that you use this macro with the MOVEC macro.
- When using the SVM address returned by this macro, do not attempt to cross a page boundary without issuing another GSVAC macro. Conversion of the EVM address for the next page will avoid errors caused by discontinuous SVM storage.
- The macro can return incorrect results if addresses passed to it have their high order bit (X'80') set. The technique of loading a zero address, such as,  
LA Rx,0(Rx)

before the macro call clears out registers in the way required.

## Examples

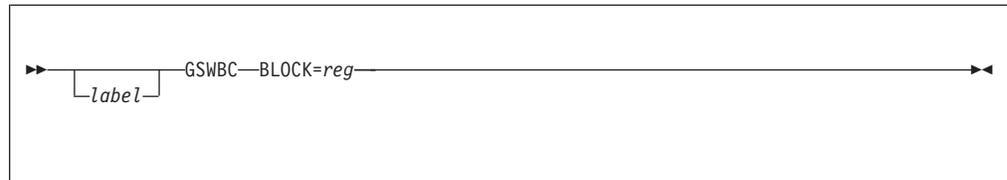
None.

## GSWBC—Get a System Work Block (SWB) Address

Use this system macro to obtain an available system work block (SWB) storage address.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

**BLOCK=reg**

Specifies where the SWB address will be placed after the macro completes processing, where *reg* is a register from R0 – R7.

### Entry Requirements

This macro is restricted to ECB-controlled programs.

### Return Conditions

None.

### Programming Considerations

- An SWB is not released when the entry control block (ECB) exits. To prevent the TPF system from running out of SWBs, the application program that issues the GSWBC macro must also release the SWB by issuing the RSWBC macro. See “RSWBC—Release a System Work Block (SWB)” on page 412 for more information about the RSWBC macro.
- The address returned is a 31-bit address and may point to an SWB allocated above the 16-MB boundary. The application program must be running in 31-bit mode to access data above the 16-MB boundary.

### Examples

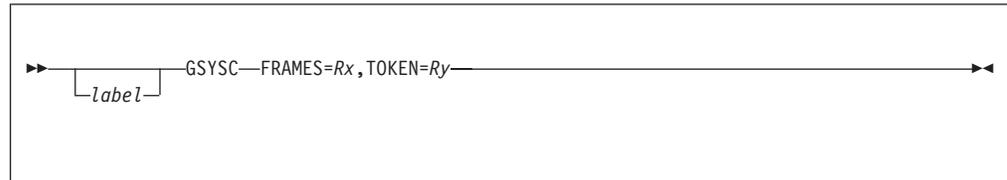
The following example gets an SWB and places the address in register 5 (R5).

```
GSWBC BLOCK=R5
```

## GSYSC—Get System Heap Storage

Use this system macro to allocate the specified number of 4 KB frames as contiguous storage in the system heap. This macro is used by applications while the \$GSYSC macro is used by the control program (CP). See “\$GSYSC—Get System Heap Storage” on page 50 for more information about the \$GSYSC macro.

### Format



*label*

A symbolic name may be assigned to the macro statement.

**FRAMES=*Rx***

The FRAMES parameter specifies the number of contiguous 4 KB frames to be allocated. The general register used must be R0 through R7 or R14.

**TOKEN=*Ry***

The TOKEN parameter specifies the address of an 8-character string that the TPF system uses to identify the allocated storage. The general register used must be R0 through R7, R14, or R15.

### Entry Requirements

R9 must contain the address of a valid ECB.

### Return Conditions

- The register specified by FRAMES contains the starting address of the allocated storage.
- The register specified by FRAMES contains zero if:
  - The FRAMES parameter register contains zero.
  - The system heap does not contain enough storage to satisfy the allocation request.
  - The address specified by the TOKEN parameter is not in the range of addressable memory.
- All registers are preserved across the macro call except for the registers used for the FRAMES and TOKEN parameters.
- Control is not returned if there is not enough real storage to satisfy the request. The TPF system issues an error and recovers through an automatic IPL.

### Programming Considerations

- Real storage may not be contiguous.
- The amount of virtual storage available for the system heap is defined in CTKA.
- Storage **must** be released by the RSYSC or \$RSYSC macro when it is no longer needed. If the storage is not released, it remains in use until you IPL the TPF system again.
- The address returned is a 31-bit address.
- The storage key is set to X'C'.

## GSYSC

- When the macro completes processing successfully, the macro trace entry contains the size (in number of frames) and address of the allocated area in addition to the normal macro trace information.

## Examples

The following example shows how the length of a block is converted into a number of 4 –KB frames before requesting storage from the system heap. The return code is checked before trying to use the address in R14.

```
      ITUUTL REG1=R14                CONNECT WITH TABLE UPDATE DSECT
      LA    R14,ITULEN                GET THE LENGTH OF A BLOCK
      LA    R14,4095(R14)             ROUND TO THE NEXT 4 KB
      LR    R7,R14                    SAVE NUMBER OF FRAMES
      SRL   R14,12                    DETERMINE NUMBER OF 4 KB FRAMES
      LA    R6,MY_TABLE
      GSYSC FRAMES=R14,TOKEN=R6        ALLOCATE THE STORAGE
      LTR   R14,R14                   CHECK THE RETURN CODE
      BZ    NO_STORAGE_AVAIL          BRANCH TO PROCESS ERROR
      .
      .
      routine that uses the storage
      .
      .
RELEASE_STORAGE DS 0H
      LA    R6,MY_TABLE
      RSYSC ADDRESS=R14,FRAMES=R7,TOKEN=R6  RELEASE THE STORAGE
      LTR   R15,R15                   CHECK THE RETURN CODE
      BNZ   RELEASE_ERROR            BRANCH TO PROCESS ERROR
      .
      .
      .
      .
MY_TABLE DC    CL8'MY_TABLE'
```

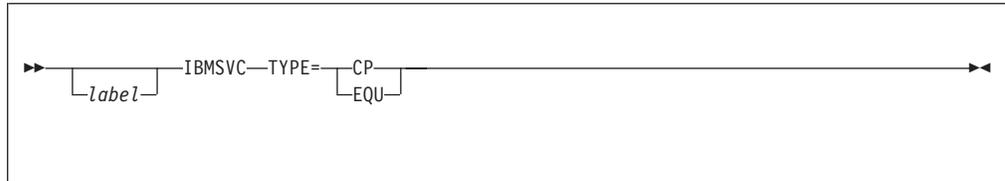


## IBMSVC—Generate IBM SVC and Fast-Link Tables

Use this system macro to generate all the IBM fast-link tables and the IBM primary and indexed supervisor call (SVC) tables. This macro also contains the CRESVC macro calls that create the IBM primary table, fast-link table, and the IBM indexed table.

See “CRESVC—Create an SVC/Fast-Link Table Entry” on page 180 for more information about the CRESVC macro.

### Format



#### TYPE

This parameter specifies whether tables or equates should be generated.

#### CP

Using this parameter creates the actual primary, indexed, and fast link tables. It is used for the table in CCMCDC.

#### EQU

Using this parameter creates a set of symbolic equates for the macro SVC, index or fast-link numbers for use in CPSEQ and other areas.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

TYPE=CP should be used only once, in the CCMCDC CSECT. TYPE=EQU should only be used in CPSEQ.

### Examples

None.

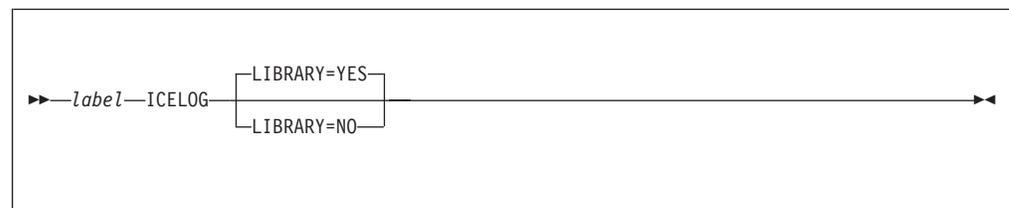
## ICELOG–TARGET(TPF) C Language Support Epilog

Use this system macro only with TARGET(TPF) support to return from a TARGET(TPF) assembler C library function or external function that begins with the ICPLOG macro by generating code to restore the registers saved by the preceding ICPLOG macro call. If a TARGET(TPF) stack frame was allocated by the ICPLOG macro, it is released by the ICELOG macro.

The ICELOG macro calls are required for writing TARGET(TPF) C library functions in assembler language and for external functions written in assembler that need to manage TARGET(TPF) stack storage.

See “ICPLOG–TARGET(TPF) C Language Support Prolog” on page 286 for more information about the ICPLOG macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **LIBRARY**

Specify one of the following:

##### **YES**

Indicates that the function is a TARGET(TPF) C library function. This is the default.

##### **NO**

Indicates that the function is an external function.

### Entry Requirements

R9 must contain the address of the current ECB.

### Return Conditions

Control is never returned to the program that issues the ICELOG macro.

### Programming Considerations

- Use this macro only in TARGET(TPF) C applications.
- This macro can be run on any I-stream.
- After the ICELOG runs, the program that called this function regains control. R7 and CE1STK will contain the address of the TARGET(TPF) stack frame that belongs to the program that called this function. The contents of R0 through the register specified by the HIGHREG parameter of the ICPLOG macro will be restored to the values they contained on entry to the function.
- This macro can be coded more than once in the same source file even though good coding practice normally encourages a single exit point from a function.

## ICELOG

- When coding this function, you are responsible for ensuring that the function's value, if any, is returned properly. There are several possibilities based on the data type of the return value:
  - R6 itself contains the value if the return value is an integer, a character, or a pointer.
  - If the return value is a float or double word, the value is accessed through a pointer contained in the first word of the parameter list.
  - ICELOG cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES). The TMSEC macro should be used instead. See *TPF General Macros* for more information about the TMSEC macro.

See “ICPLOG–TARGET(TPF) C Language Support Prolog” on page 286 for examples.

## Examples

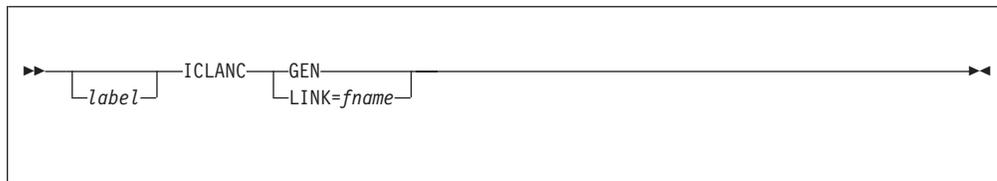
None.

## ICLANC—Call a Secondary Library Routine

Use this system macro with C language support to:

- Produce a table of V-type address constants (VCONs) that contain the entry points to the secondary library routines. This option is valid only in the control program (CP) and is restricted to the CCLANG CSECT.
- Generate linkage to a particular secondary library routine from its corresponding TPF-written library function.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **GEN**

This option is coded in the C language support CSECT (CCLANG) to generate a table of VCONs used to link to the secondary library routines.

#### **LINK=fname**

The name of a service routine using MVS-style parameter and register conventions. Code is produced to save registers, convert to IBM MVS-style register conventions, generate the quick enter linkage for the secondary library routines, and to convert back to TPF register conventions on return.

### Entry Requirements

R9 must contain the address of the current ECB.

### Return Conditions

TPF register contents are restored. Registers R10 through R13 remain unchanged.

### Programming Considerations

This macro can be run on any I-stream.

### Examples

This example generates the table of VCONs for secondary library routines:

```
ICLANC GEN
```

This example generates linkage to the secondary library routine called by the `sin` (sine) function:

```
ICLANC LINK=SIN
```

## ICPLOG–TARGET(TPF) C Language Support Prolog

Use this system macro with TARGET(TPF) C language support to allocate a TARGET(TPF) stack frame for the function from the current TARGET(TPF) stack block. If there is no current TARGET(TPF) stack block or if the requested stack frame does not fit in the current block, the TARGET(TPF) stack exception routine in the control program (CP) is activated.

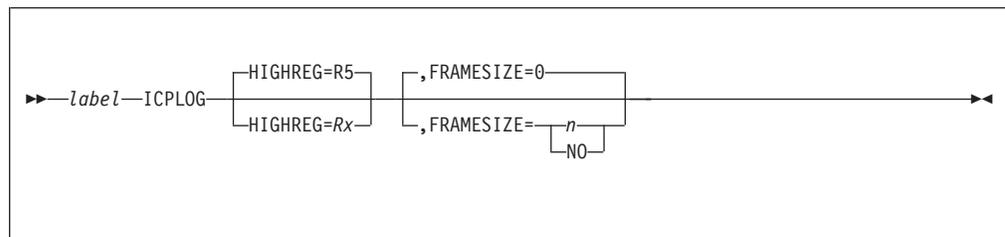
This macro is required for writing TARGET(TPF) C library functions in assembler language and for assembler external functions that need to manage TARGET(TPF) stack storage. It must be the first instruction coded after the BEGIN macro. See *TPF General Macros* for more information about the BEGIN macro.

The ICPLOG macro generates code that stores specified registers in the:

- Stack frame of the TARGET(TPF) calling program and saves the calling program's R14 used as the return linkage register by library functions
- Calling program's TARGET(TPF) C language stack frame, and saves the calling program's R14 used as the return linkage register.

The ICPLOG macro allocates a TARGET(TPF) stack frame for the function from the current TARGET(TPF) stack block. If there is no current TARGET(TPF) stack block or if the requested stack frame does not fit in the current block, the TARGET(TPF) stack exception routine in the control program (CP) is activated.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **HIGHREG=reg**

The prolog stores registers R14-**reg** into the caller's TARGET(TPF) stack frame, where **reg** is in the R0–R5 range. The default is R5.

#### **FRAMESIZE**

Specify one of the following:

- 0** Indicates that the function does not use the TARGET(TPF) C language stack for its own purposes, but calls another function that requires a TARGET(TPF) stack frame to save and restore registers and the return z address. ICPLOG allocates the minimum size stack frame. This is the default.
- n* The number of bytes of storage in the new TARGET(TPF) stack frame to be reserved for the function's use, which is added to the size of the minimum stack frame.
- NO** Indicates no stack frame is to be allocated.

## Entry Requirements

R9 must contain the address of the current ECB.

## Return Conditions

- R7 and CE1STK will contain the address of the newly acquired TARGET(TPF) stack frame unless FRAMESIZE=0 is coded.
- R14 and the registers requested by the macro call are saved in the caller's TARGET(TPF) stack frame.

## Programming Considerations

- Use this macro only in TARGET(TPF) C language applications.
- This macro can be run on any I-stream.
- If there is no TARGET(TPF) stack block when the macro is called, the stack exception routine is called, which acquires a TARGET(TPF) stack block and initializes it. If the stack exception routine is unable to acquire a stack block, the appropriate dump is issued and the ECB is exited.
- Ensure that the minimum stack frame size plus the size requested by the FRAMESIZE parameter do not exceed the maximum TARGET(TPF) stack frame size. See data macro ICS0TK for the equates corresponding to minimum (CSTKMIN) and maximum (CSTKFRSZ) stack frame size.
- ICPLOG should be issued only once in a given source module.
- ICPLOG cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES). Use the TMSPC macro instead.

## Examples

This example creates a prolog that saves registers R14 through R5 and does not change the size of the function stack frame. R5 is required by the TIMEC macro. The registers are set up with R5 pointing at the stack frame and R8 operating as the application base register. The C library base previously in R8 is saved. After the TPF macro returns, the C library base is restored and the address of the timestamp provided by TIMEC is placed in the return register (R6) for TARGET(TPF) C. The ICELOG macro stores the C environment and returns to the calling C program.

```
ICPLOG HIGHREG=R5,FRAMESIZE=NO      R5 needed for TIMEC interface
L   R5,CE1TCA                       points to first stack frame
SL  R5,=A(CSTKTCA-ICS0TK)           compute base of first frame
L   R5,CSTKUEXP-ICS0TK(,R5)         get address of user exp. area
ST  R8,CSTKLBAS                     save library base
L   R8,CE1SVP                       get application base
TIMEC ,                             get the timestamp
L   R8,CSTKLBAS                     restore library base
LR  R6,R5                           return the address in R6
ICELOG ,                             return to C application
```

The following example creates a prolog that saves registers R14 through R3 and reserves 66 bytes of storage in the stack frame for the function:

```
ICPLOG HIGHREG=R3,FRAMESIZE=66
```

The following example creates a prolog that saves registers R14 through R2, and allocates a minimal stack frame:

```
ICPLOG HIGHREG=R2,FRAMESIZE=0
```

The following example creates a prolog that saves registers R14 through R5 (because R5 is the default), and reserves 212 bytes of storage in the stack frame:

## ICPLOG

```
ICPLOG FRAMESIZE=212
```

The following example creates a prolog that saves registers R14 through R5, and does not allocate a stack frame:

```
ICPLOG FRAMESIZE=0
```

The following example has the same effect as the previous one — it creates a prolog that saves registers R14 through R5, and does not allocate a stack frame (because FRAMESIZE=0 is the default):

```
ICPLOG
```

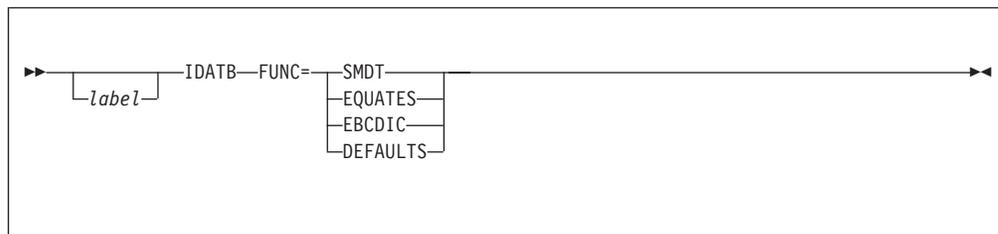
The following example is a TARGET(TPF library function that calls a TPF macro to put a time stamp in the stack frame. The function returns a pointer to the time stamp as its value.

```
        PRINT NOGEN
        BEGIN NAME=C001,VERSION=31
*
        ICPLOG HIGHREG=R5,FRAMESIZE=NO      R5 needed for TIMEC interface
        L      R5,CE1TCA                    points to first stack frame
        SL     R5,=A(CSTKTCA-ICS0TK)        compute base of first frame
        L      R5,CSTKUEXP-ICS0TK(,R5)      get address of user exp. area
        ST     R8,CSTKLBAS                  save library base
        L      R8,CE1SVP                    get application base
        TIMEC ,                             get the timestamp
        L      R8,CSTKLBAS                  restore library base
        LR     R6,R5                        return the address in R6
        ICELOG ,                             return to C application
*
        LTOrg
        FINIS
        END
```

## IDATB—Build Selective Memory Dump Table

Use this system macro as the repository for all information about the selective memory dump table (SMDT). This macro contains a set of IDATG macro calls, each of which defines a unique parameter and its associated main storage area. The IDATB macro can be used anywhere in the TPF system to obtain information about the large data areas defined to the TPF system that are not normally included in a dump of system storage.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FUNC**

One of four values can be assigned:

##### **SMDT**

This option is used by copy member CPST of the CCCPSE CSECT to actually construct the selective memory dump table in main storage.

##### **EQUATES**

This option is used by the IDOTB macro and the system initializer (CCCTIN) to generate a set of equates, one for each parameter. It is used by IDOTB so that the assembler will recognize the values specified for the INCLUDE parameter. It is used by the system initializer (CCCTIN) to access the correct SMDT entry for a specific parameter so that the starting and ending address pairs for a particular parameter can be stored in the entry.

##### **EBCDIC**

This option is used by the ZIDOT command processor to construct a table linking EBCDIC values to SMDT index values and uniqueness indicators. The table is mapped by the IDSEBC macro.

##### **DEFAULTS**

This option is used by the ZIDOT command processor to construct a default bitmap with bits already set on for values defined as REQUIRED=YES on the IDATG macro calls.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- Additions to or deletions from the list of IDATG macro calls in the IDATB macro will cause any ZIDOT overrides that are stored on the online system to be

## IDATB

deleted when the TPF system is IPLed. When items are added or deleted, you clear the existing overrides using the ZIDOT CLEAR command. Care should be exercised when changing DUMMY entries, because there can be hidden dependencies on them.

- The following programs must be reassembled whenever IDATG macro calls are added, removed, or changed in any way in the IDATB macro.

When Updating	Reassemble	Link Edit
IDATB/UDATB	CCCTIN CCUEXT ICDF STPP	CPS0   PPCP
CIDP	CCCPSE	CPS0
CUDP	CCCPSE CCUEXT	CPS0

- The IDATB calls required by the TPF system are already in place. Normally you do not need to code the IDATB macro explicitly.
- User-defined parameters are defined in the UDATB copy member, which is called by the IDATB macro.
- The following parameters are predefined. All parameters defined by IBM begin with the character I; otherwise, the parameter can begin with any alphabetic character.

Parameter	Description
IACN	Current Activation Number
IAET	RCS Asynchronous Event Table
ICAPT	Low Memory (CAPT)
ICCT	Communication Configuration Table
ICILT	Internal Line Number Table
ICIOLDV	Common I/O (CIO) Idevbks
ICIOLIT	CIO Lost Interrupt Tables
ICIOLMP	CIO Logical Device Map
ICLAW	CLAW Tables
ICLH	CLH Block Management Tables
ICNFTBL	BSS CINFC Table
ICOMMON	4 K Common Blocks
ICORND	All of Memory
ICP	Control Program (CP)
ICPCM	CCP Common Area
ICPLC	CCP Low Core Trace Blocks
ICPLKMP	CP Linkage Map
ICPSE	CPSE Internal Entry Trace

ICRH	31-bit Core Resident Program Area (CRPA)
ICRPGM	24-bit Core Resident Program Area (CRPA)
ICTK2	SNA Keypoint/ncst
IDCLS	CLH Dispatch Control Lists
IDDMFDO	TFPAR DDM FDOCA Tables
IEAT	ECB Activation Table
IECBS	Entry Control Blocks (ECBs)
IELDR	E-Type Loader Indicators
IEPOL	E-Type Loader Policing Values
IEVM DAT	ECB Virtual Memory (EVM) Page and Segment Tables
IFACE	File Address Compute Program (FACE) Table
IFRM	4 K Frames
IGAT	Global Attribute Table
IGLBLI	High Storage Primary Globals
IIOBS	I/O Control Blocks
IISG1	I-Stream (IS)-Shared Global Area 1
IISG2	I-Stream (IS)-Shared Global Area 2
IISG3	I-Stream (IS)-Shared Global Area 3
IISG4	I-Stream (IS)-Shared Global Area 4
IIUG1	I-Stream (IS)-Unique Global Area 1
IIUG2	I-Stream (IS)-Unique Global Area 2
IIUG3	I-Stream (IS)-Unique Global Area 3
IIUG4	I-Stream (S)-Unique Global Area 4
ILCPT	Low Core Patch Area
ILRT	Link Routing Table
IMFST	Module File Status Table
IMPIF	Multi-Processor Interconnect Facility (MPIF) Control Tables
IPAT	Program Allocation Table (PAT)
IPATHSH	PAT Hash Table
IPATIST	IS-Unique PAT Entries
IPATXTP	E-Type Loader Extra PAT Entries
IPLMT	Pool Management Table
IPOOL	Pool Directories
IRCSB	Record Cache Subsystem (RCS) Subsystem Status Table Base
IRCSH	RCS Subsystem Status Table Header
IRCSO	RCS Subsystem Status Table Overflow
IRIAT	Record ID Attribute Table (RIAT)

## IDATB

ISMDT	Selective Memory Dump Table (SMDT)
ISNA	System Network Architecture (SNA) Tables
ISNL	SNA Local Element Bit Map
ISNPT	System Ordinal Number (SON) Pointers
ISNTB	SON Tables
ISVMDAT	System Virtual Memory (SVM) Page and Segment Tables
ISWBS	System Work Blocks (SWBs)
ITAR	TPFAR Tables
ITCP	TCP/IP Native Stack Support Tables
IUSREXT	User Exit Table
IVFABUF	Virtual Vile Access (VFA) Buffers
IVFA1	VFA RSHT Area
IVFA2	VFA RSTBCA Area
IWGTA	WGTA Tables
IWRT	E-Type Loader Working Record Table
IXGLB1	Extended Global Area 1
IXGLB2	Extended Global Area 2
IXGLB3	Extended Global Area 3
IXGLB4	Extended Global Area 4
IXID	Exchange Identification (XID) I/O Trace Table.

## Examples

- Build the Selective Memory Dump Table (SMDT)  
IDATB FUNC=SMDT
- Build the EBCDIC keyword table  
IDATB FUNC=EBCDIC
- Make keyword equates available  
IDATB FUNC=EQUATES
- Define the default bit map  
IDATB FUNC=DEFAULT

## IDATG—Generate Selective Memory Dump Table Entry

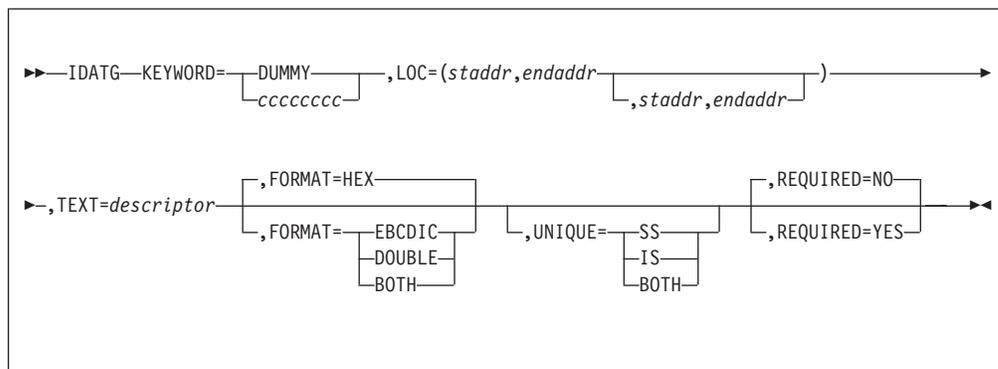
Use this system macro to to:

- Map a user-defined parameter to a particular region of main storage. The parameter identifies main storage regions for inclusion in dumps through the IDOTB macro, the ZIDOT command, or in every dump (REQUIRED=YES).
- Identify very large data areas that are not needed in every dump. The areas identified by the IDATG macro will not be dumped, unless a dump override is created for a particular system error that requests that a storage area be included.

This macro is only coded within the IDATB or UDATB macros. It is never used in open code. Each IDATG call results in the creation of a selective memory dump table (SMDT) entry in the CPST copy member of the CCCPSE CSECT.

See “IDOTB—Dump Override Table Build” on page 296 for more information about the IDOTB macro. See “IDATB—Build Selective Memory Dump Table” on page 289 for more information about the IDATB macro. See *TPF Operations* for more information about the ZIDOT command.

### Format



#### KEYWORD

Required. Specify one of the following:

*keyword*

Identifies the area of main storage. The value specified can be no longer than 8 alphanumeric characters, and the first character must be a letter.

#### DUMMY

This form of the macro generates a dummy SMDT entry that will be available for use when creating temporary keywords using the ZIDOT command. The number of `KEYWORD=DUMMY` calls determines how many temporary keywords can be created. If `DUMMY` is coded no other parameters are allowed.

#### LOC=(staddr,endaddr [...])

Required. The starting address and ending address of the storage area. If the area resides in the control program (CP), an address may be specified as either a VCON or an ADCON. If the area resides outside of the control program (CP), the addresses must be specified as ADCONS - `A(0),A(0)` - and code must be added to `CCCTIN` to store the starting and ending addresses of the storage

## IDATG

area in the Selective Memory Dump Table Entry. Multiple pairs of starting/ending addresses may be specified, unless the UNIQUE parameter is coded.

**Note:** If WXTRNs are required they must be included in the IDATB macro. See "IDATB–Build Selective Memory Dump Table" on page 289 for more information about the IDATB macro.

### **TEXT=***descriptor*

Required. A text description of the storage area. This is displayed in the dump, and in ZIDOT command responses. The description is limited to 36 characters, and must not contain control characters.

### **FORMAT**

Optional. One of four possible values which specifies how the data in the area of storage being defined should be formatted in the dump. Valid parameters are:

#### **HEX**

Format the data in hex.

#### **EBCDIC**

Convert the data to printable characters.

#### **DOUBLE**

Format data in hex, followed by a second line in EBCDIC.

#### **BOTH**

Format data in both hex and EBCDIC on the same print line. Causes printing of dump labels to be suppressed.

The default is FORMAT=HEX.

### **UNIQUE**

Optional. Specifies that multiple copies of the storage area exist, as follows:

#### **SS**

Generate multiple selective memory dump table entries for this keyword, one for each subsystem.

**IS** Generate multiple selective memory dump table entries for this keyword, one per I-stream.

#### **BOTH**

Generate multiple selective memory dump table entries for this keyword, one for each I-stream and subsystem.

There is no default.

**Note:** This keyword cannot be coded if control program (CP) VCONs are used to identify the storage area or if more than one pair of start/end addresses was coded on the LOC parameter.

### **REQUIRED**

Optional. Specify one of the following:

#### **YES**

This storage area is unconditionally included in all dumps.

#### **NO**

The area only appears in a given dump if an override has been created for the dump via the IDOTB macro or ZIDOT command.

The default is REQUIRED=NO.

## Entry Requirements

None.

## Return Conditions

None.

## Programming Considerations

The IDATG calls need not be coded in ascending address order. The initializer program will sort the SMDT entries according to whether the data areas are present in the user's configuration.

## Examples

```
IDATG KEYWORD=I-CP,           the control program
      FORMAT=HEX,             format in hex
      LOC=(V(CPMEXR),V(CPLKM2)), start/end addresses
      TEXT='CONTROL PROGRAM'  text for display

IDATG KEYWORD=DUMMY           reserve some dummy entries
IDATG KEYWORD=DUMMY
IDATG KEYWORD=DUMMY

IDATG KEYWORD=I-MFST,         Module File Status Table
      FORMAT=BOTH,            format hex and EBCDIC
      LOC=(A(0),A(0)),        CCCTIN will initialize
      TEXT='MODULE FILE STATUS TABLE', text for display
      UNIQUE=SS               TPF subsystem unique
```

**Note:** All parameters defined by IBM begin with the character I.

## IDOTB–Dump Override Table Build

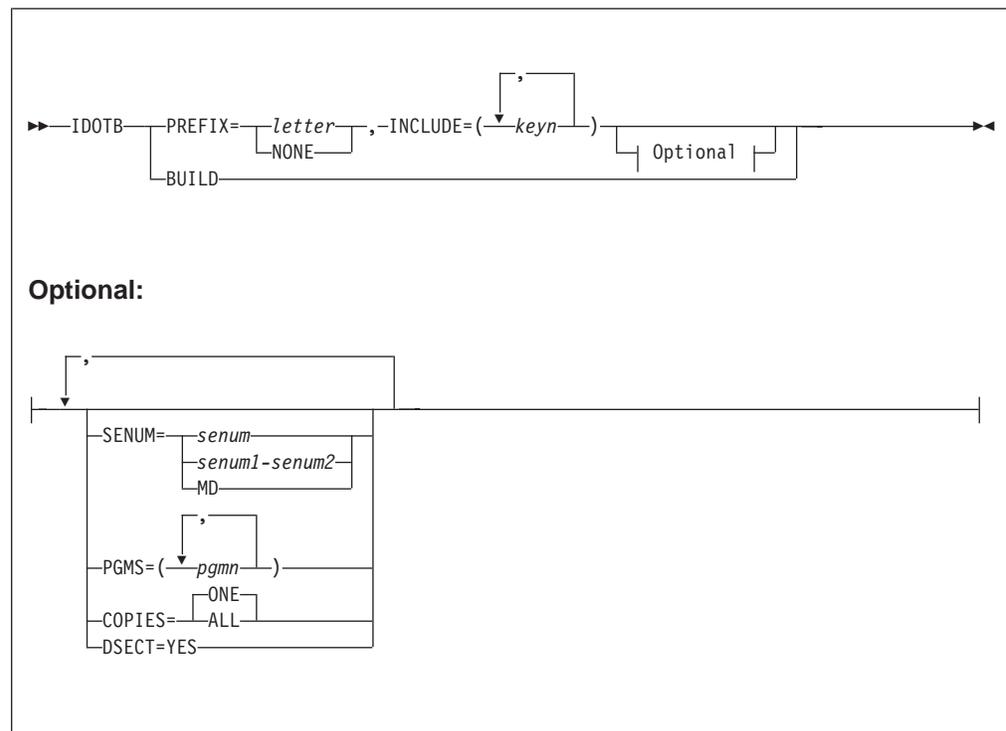
Use this system macro to identify main storage areas for inclusion in a dump for a particular system error or range of system errors. These areas are in addition to those specified in the SLIST parameter of the SERRC macro. The areas are identified by means of a bitmap.

Each IDOTB macro call generates an entry in a portion of the dump override table called the static override bitmap table (SOBT). The dump overrides are defined by means of a bitmap. There are two sets of IDOTB macro calls in the TPF system:

- The CIDP copy member of the CCCPSE CSECT contains the dump overrides for IBM system errors
- The CUDP copy member of the CCUEXT CSECT contains the dump overrides for user system errors.

See *TPF General Macros* for more information about the SERRC macro.

## Format



### PREFIX

This parameter is required. Specify one of the following:

#### letter

An uppercase alphabetic character, coded on the SERRC macro, that is concatenated with the system error number in the console message and in the dump. The letters I and W through Z are reserved for IBM use.

#### NONE

The designated override will take effect only if no prefix appears in the SERRC macro expansion; that is, the SERRC macro call resulted from a C language perror or exit function call.

**SENUM=*senum* | (*senum1-senum2*) | MD**

The six-digit system error number used to identify the dump. The values specified may be coded two ways:

- A symbolic system error equate; for example, CA9T51.
- A self-defining hexadecimal value; for example, X'123456'.

Use MD to define the overrides for a manual dump. If this parameter is coded, the override will only take effect if the system error number for the error matches the value specified, or falls within the range *senum1-senum2*.

**Note:** No check is made to ensure that *senum2* is greater than *senum1*.

**PGMS=(*pgm1* [,...,*pgmn*])**

One or more ECB-controlled program names to be associated with this override. If this parameter is coded, the override will only take effect if the system error was issued by one or more ECB-controlled programs specified.

Each program name must consist of a four-character segment name and a 2-character version code. A version code of \*\* will cause the override to be in effect for all program versions.

**Notes:**

1. Due to a limitation of the macro language, only 255 characters may be input per macro call. If more program names are required, code multiple IDOTB macros.
2. Overrides having an exact match on the failing program version code will take priority over those coded with \*\*.

**INCLUDE=(*key1* [,...,*keyn*])**

The selective memory dump table (SMDT) values that identify the areas of main storage to be included. See "IDATB—Build Selective Memory Dump Table" on page 289 for a list of valid values. This parameter is required.

Because of macro language limitations, only 255 characters may be coded. If you want more characters, use the INCLUD2 parameter.

**INCLUD2=(*key1* [,...,*keyn*])**

This handles the overflow from the INCLUDE parameter. See the INCLUDE parameter for more information.

**COPIES**

A single copy or multiple copies of a storage area can be specified, when there are multiple copies for each I-stream or MDBF subsystem.

**ONE**

Only the copy for the subsystem or I-stream where the error occurred will appear in the dump.

**ALL**

All copies of the storage area will appear in the dump.

The default is COPIES=ONE.

**Note:** This parameter also applies to storage areas defined using more than one pair of start/end addresses on the IDATG macro.

**DSECT=YES**

This form of the macro is used to generate DSECTs and equates describing the SOBT data structure. It is intended for CP use only.

## IDOTB

### BUILD

This parameter must be coded by itself and is used to identify the end of the set of IDOTB macro calls. It causes the previous calls to be validated, and if no errors are detected, the static override table will be built.

## Entry Requirements

None.

## Return Conditions

None.

## Programming Considerations

- See “IDATB–Build Selective Memory Dump Table” on page 289 for more information about valid parameters.
- The following priority sequence determines which overrides apply to an error (from highest to lowest):
  1. An override specifies a system error number and a program name list.
  2. An override specifies a range of system error numbers and a program name list.
  3. An override specifies a system error number.
  4. An override specifies a range of system error numbers.
  5. An override specifies a program name list, but no system error numbers.
  6. An override specifies neither a program name list nor system error numbers. (for example, the prefix character only).

## Examples

The operands on multiple IDOTB calls may overlap. When this occurs, system error processing will determine which override to use based on an established priority. Consider the following:

- IDOTB PREFIX=Q,INCLUDE=(I-MFST)  
Include the MFST in the dump for all system errors having a prefix of Q.
- IDOTB PREFIX=Q,PGMS=(CVFE\*\*,CNPU\*\*),INCLUDE=(I-GAT)  
Include the GAT (and not the MFST) if the system error has a prefix of Q and the SERRC was issued by programs CVFE or CNPU.
- IDOTB PREFIX=Q,SENUM=(X'101'-X'1FF'),INCLUDE=(I-MPIF,I-LSWB)  
Include the MPIF control tables and SWBs (and not the GAT or MFST) if the system error number has a prefix of Q and the system error number falls within the range shown.
- IDOTB PREFIX=Q,SENUM=X'1F0',INCLUDE=(I-CIOLDV)  
Include the CIO LDEV blocks (and none of the previous areas) if the system error has a prefix of Q and the system error number is X'1F0'.  
  
**Note:** This falls in the range X'101' through X'1FF' specified on the previous IDOTB call.
- IDOTB PREFIX=Q,SENUM=X'1F0',PGMS=(CYEA\*\*,CYEM\*\*),INCLUDE=(I-LIOCB)  
Include the IOCBs (and none of the previous areas) if the system error has a prefix of Q, the system error number equals X'1F0', and the SERRC was issued by program CYEA or CYEM.

In these situations system error processing will identify all of the overrides that may apply to the error. Then an override is selected using the priority scheme discussed previously.

- When the same priority level is defined by two or more applicable overrides, system error processing will use the first override found in the table.

XYZ	IDOTB BUILD	construct the SOBT
XYZ	IDOTB DSECT=YES	generate the SOBT DSECTs

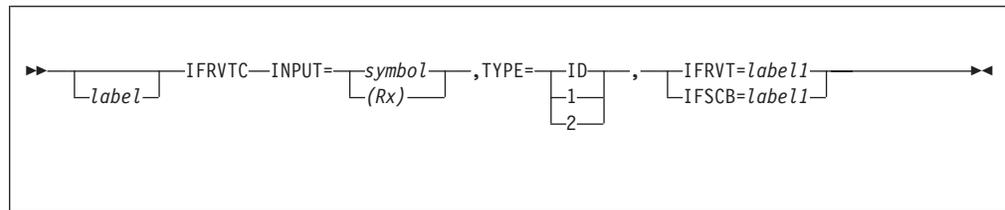
## IFRVTC–Test RID/RVT Address

Use this system macro to distinguish between:

- A resource identifier (RID) and an SCBID
- An RVT1 address and an SCB1 address
- An RVT2 address and an SCB2 address.

This macro causes a conditional branch depending on whether the tested value refers to a resource vector table (RVT) or a session control block (SCB).

### Format



#### INPUT

This parameter specifies the location of the input to be tested.

##### *symbol*

The symbolic address of the input. If the input is an RID or an SCBID, *symbol* is a 3-byte field; if the input is an RVT1, RVT2, SCB1, or SCB2 address, *symbol* is a 4-byte field.

##### *(Rx)*

The register containing the input. If the input is an RID or an SCBID, it is right-justified in *(Rx)* with the high-order bits set to zeros; if the input is an RVT1, RVT2, SCB1, or SCB2 address, *(Rx)* contains the 4-byte address. Valid registers are R0–R7 and R15.

#### TYPE

This parameter specifies the type of input.

**ID** The input is either an RID or an SCBID.

**1** The input is either an RVT1 address or an SCB1 address.

**2** The input is either an RVT2 address or an SCB2 address.

##### **IFRVT=label1**

This parameter specifies the symbolic label to which control is transferred if the input refers to an RVT.

##### **IFSCB=label1**

This parameter specifies the symbolic label to which control transferred if the input refers to an SCB.

### Entry Requirements

- The input value must be a valid RID, SCBID, RVT1 address, SCB1 address, RVT2 address, or SCB2 address, corresponding to the value of the TYPE parameter.
- Specify only the IFRVT parameter or the IFSCB parameter; do not specify both.
- If the input is specified as a register, it must be one of R0 through R7 or R15, enclosed in parentheses.

- All labels specified as operands must be addressable using a register other than R14.

## Return Conditions

- If the input references an RVT and a label is coded with the IFRVT parameter, a branch is taken to the specified label; otherwise, processing continues following the macro expansion.
- If the input references an SCB and a label is coded with the IFSCB parameter, a branch is taken to the specified label; otherwise, processing continues following the macro expansion.
- The contents of R14 are unknown. All other registers are unchanged.

## Programming Considerations

- This macro is for use by systems programs only.
- This macro does not validate the input value.

## Examples

```

INPUT_RID_OR_SCBID EQU EBR0UT
.
.
.

IFRVTC INPUT=INPUT_RID_OR_SCBID,TYPE=ID,IFSCB=FLAG_SCBID
MVI ID_FLAG,IS_A_RID
B NOW_WE_KNOW
FLAG_SCBID DS 0H
MVI ID_FLAG,IS_AN_SCBID
NOW_WE_KNOW DS 0H
.
.
.

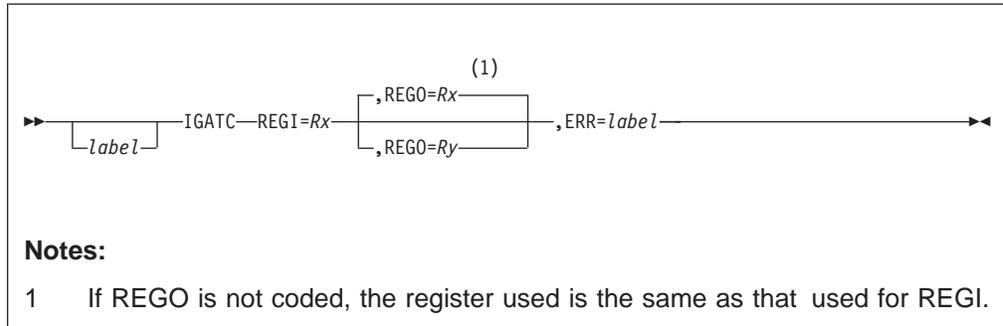
IFRVTC INPUT=(R2),TYPE=1,IFRVT=ITS_AN_RVT1_ADDRESS
BAS R7,PROCESS_SCB1
B RESOURCE_HAS_BEEN_PROCESSED
ITS_AN_RVT1_ADDRESS DS 0H
BAS R7,PROCESS_RVT1
RESOURCE_HAS_BEEN_PROCESSED DS 0H
.
.
.

```

## IGATC—Get Global Attribute Table Entry

Use this system macro to obtain addressability for the attributes of globals.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### **REGI=R<sub>x</sub>**

Input register, required. REGI must specify a register from R14 through R7 that contains the address of a global directory slot. For C-type programs only, you must specify R1.

#### **REGO=R<sub>x</sub>**

Output register, optional.

If coded, you must specify a register from R14 through R7 that will contain the address of the corresponding global attribute table entry.

If not coded, REGI will be used for both input and output. If the TPF system does not contain extended globals, the value returned in REGO will be the same as the value received in REGI. In C-type programs this parameter is not needed because R1 is always used.

#### **ERR=*label***

Error routine, optional.

If coded, must specify a label where control will be transferred if the contents of REGI are not valid.

## Entry Requirements

- REGI must contain the address of a global directory slot.
- For C-type programs only: R0 must contain the SSU ID in bytes 0-1 and the I-stream number in bytes 2-3.

## Return Conditions

- The address of the corresponding Global Attribute Table entry will be returned in the output register. If there are no extended globals in the TPF system, the output register will be set equal to the input register.
- For C-type programs only, the address of the corresponding Global Attribute Table entry for the main I-stream is returned in R2.

## Programming Considerations

- This macro obtains the address of the attributes of any global in any system. Its use is optional in systems that do not contain extended globals.
- This macro is intended for use by system programs only. Application programs have no need for the information contained within the indicator byte associated with each global record.

Application programs should normally use GLOUC, KEYUC, or FILKW to request keypointing of global records and fields.

## Examples

```
IGATC REGI=R1
```

This invocation specifies that the input register is R1. The output register defaults to R1 as well.

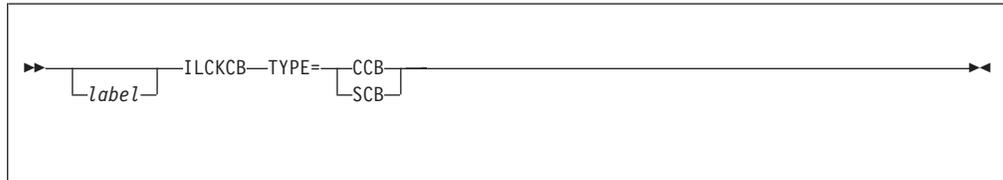
```
IGATC REGI=R14,REGO=R4,ERR=GATERR
```

This invocation specifies that the input register is R14 but the output register is R4. C-type programs cannot use this invocation, since they are limited to R1. If the contents of R14 are invalid, control is transferred to the label GATERR.

## ILCKCB—Lock a Control Block Area Macro

Use this system macro to allow an entry control block (ECB) to gain exclusive control of a control block area.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **TYPE**

The name of the control block area to be locked.

#### **CCB**

The TPF/APPC conversation control block (CCB) area is to be locked.

#### **SCB**

The TPF/APPC session control block (SCB) area is to be locked.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The calling segment must have write access (using CINFC W) to the control block to be locked.

### Return Conditions

- Control is returned to the next sequential instruction (NSI) after the control block is locked.
- After the control block is locked, the lock field in the control block contains the address of the ECB that holds this lock.
- The contents of R14 and R15 are destroyed across this macro call. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- You can run this macro on any I-stream.
- Control must not be given up between a ILCKCB macro and a IULKCB macro. Use the IULKCB macro to unlock the control block that was locked by the ILCKCB macro.
- If the control block is already held by the issuing ECB, the system error routine issues a dump with return.
- If the control block to be locked is currently held by another ECB, the code spins, waiting for the lock to be freed. If the lock is not freed, the ECB may be canceled by a timeout dump.
- Any other lock should not be held by this ECB since a lock-out condition could occur.
- This macro is restricted to real-time programs, or CP programs with a valid ECB in R9.

## Examples

```
ILCKCB TYPE=CCB
```

This invocation requests that the conversation control block (CCB) be locked.

## INDEX–Recoup Descriptor Record Structure

Use this system macro to do the following:

- Describe the location of embedded record references and the method to chain chase them.
- Code recoup descriptor container records (BKDs).

This macro is used both offline and online with the GROUP macro. INDEX macros are associated with the GROUP macro that immediately precedes them.

The offline process uses the INDEX and GROUP macros to create recoup descriptor container records.

The DSECT part of the GROUP and INDEX macros is used to access the different fields in the BKD record online.

See *TPF Database Reference* for more information about online recoup. See “GROUP–Recoup Descriptor Record Access” on page 264 for more information about the GROUP macro.

The following table shows the specification of the TYPE parameter and the requirements of the remaining parameters used with the INDEX macro.

*Table 6. Specification of the TYPE Parameter for the INDEX Macro and Parameter Requirements*

Parameter	TYP=F	TYP=C	TYP=V	TYP=L	TYP=M	TYP=S
ACODE	Optional	Error	Optional	Error	Optional	Ignored
ALTID	Optional	Optional	Optional	Optional	Optional	Ignored
BCH	Ignored	Optional	Ignored	Ignored	Ignored	Ignored
CNT	Ignored	Ignored	Optional	Optional	Optional	Ignored
CODE	Optional	Optional	Optional	Optional	Optional	Optional
CREATE	Ignored	Ignored	Optional	Optional	Optional	Ignored
DSCR	Optional	Error	Optional	Optional	Optional	Ignored
DUPEELIM	Optional	Optional	Optional	Optional	Optional	Ignored
FA	Required	Optional	Optional	Ignored	Optional	Ignored
FAT	Optional	Optional	Optional	Ignored	Optional	Optional
FCOD	Ignored	Ignored	Optional	Optional	Optional	Ignored
FI	Ignored	Ignored	Required	Required	Required	Ignored
FII	Ignored	Ignored	Optional	Ignored	Optional	Ignored
FVN	Ignored	Ignored	Ignored	Ignored	Ignored	Optional
ID	Optional	Optional	Optional	Optional	Optional	Ignored
LI	Ignored	Ignored	Required	Required	Required	Ignored
LII	Ignored	Ignored	Optional	Ignored	Optional	Ignored
LSI	Ignored	Ignored	Optional	Optional	Optional	Ignored
MAC	Optional	Optional	Optional	Optional	Optional	Ignored
NAB	Ignored	Ignored	Optional	Optional	Optional	Ignored
ORD	Ignored	Ignored	Ignored	Required	Ignored	Ignored
RCC	Optional	Optional	Optional	Optional	Optional	Ignored

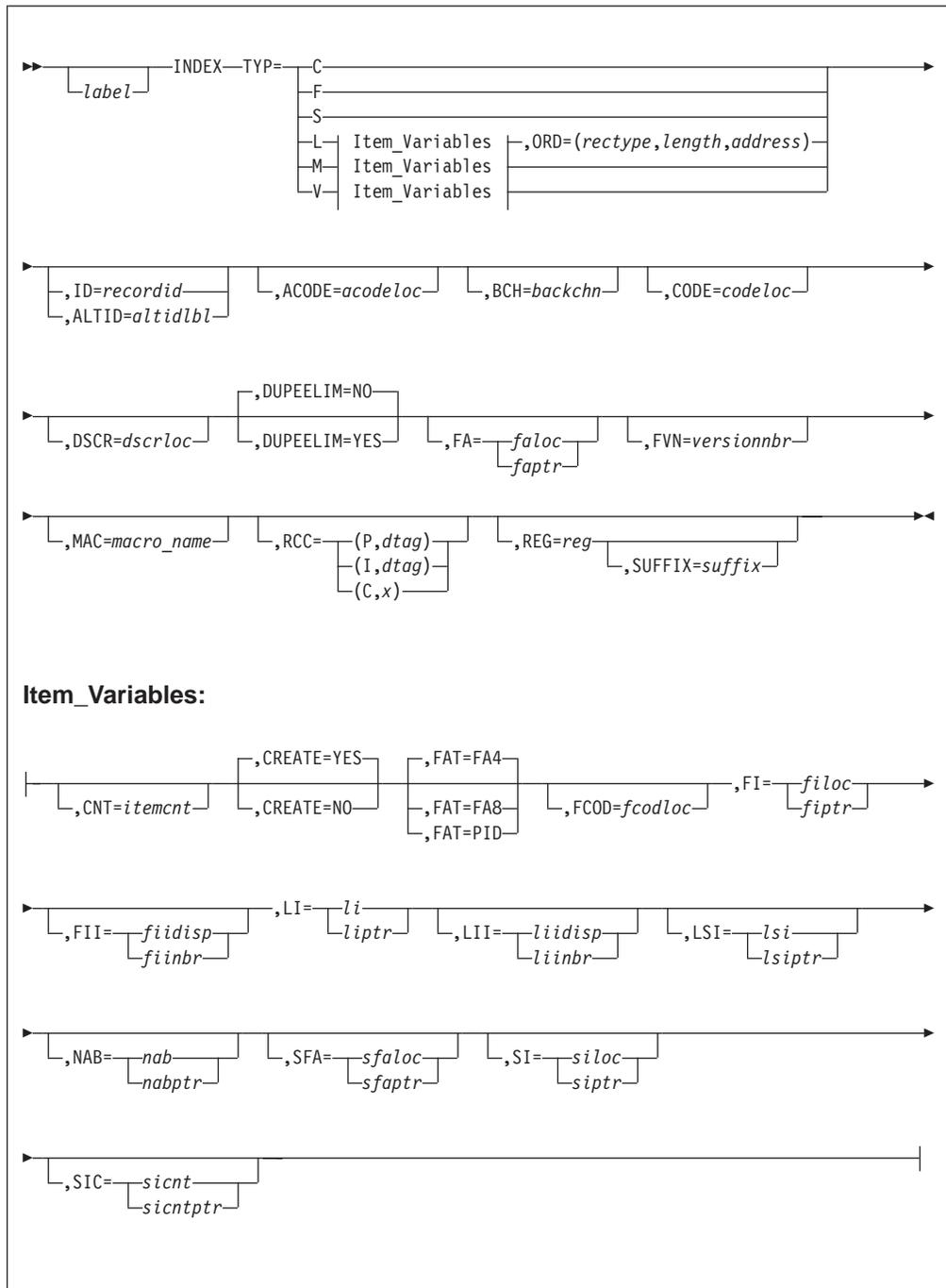
Table 6. Specification of the TYPE Parameter for the INDEX Macro and Parameter Requirements (continued)

Parameter	TYP=F	TYP=C	TYP=V	TYP=L	TYP=M	TYP=S
REG	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored
SFA	Ignored	Ignored	Optional	Optional	Optional	Ignored
SI	Ignored	Ignored	Optional	Optional	Optional	Ignored
SIC	Ignored	Ignored	Optional	Optional	Optional	Ignored
SUFFIX	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored

The following identifies the symbolic notation found in the previous table and also in the syntax information that follows.

- Error**           Parameter is not applicable and a message will be produced.
- Ignored**         Parameter is ignored (not required).
- Required**        Parameter is required.
- Optional**        Parameter is not required.

# INDEX Format



*label*

is a symbolic name that can be assigned to the macro statement.

## TYP

specifies the type of file.

- C** specifies a standard forward chaining file, where the file address is contained in a standard header (at location 8). If the file address is not at location 8, code the FA parameter.

- F** specifies a fixed address file, where the file address or persistent identifier (PID) is at a fixed location. (Do not use for forward chaining.)
- S** specifies that the file is a TPF Database Facility (TPPDF) file described by DBDEF macro statements and chain chased by TPDFDF recoup.
- L** specifies an embedded ordinal number file, where a group of ordinal numbers are described.
- M** specifies a variable item file, where file addresses are contained in variable-length items or subitems.
- V** specifies a fixed item file, where file addresses or persistent identifiers (PIDs) are contained in fixed-length items, subitems, or PIDs.

**ORD**

specifies the ordinal description if chained records contain an ordinal number, where:

*rectype*

The file address compute (FACE) program record type used to convert the ordinal number to a file address.

*length*

The length of the field containing the ordinal number.

*address*

The location relative to the first item (FI) of the ordinal number; for example:

```
ORD=(#PNDRI,1,TAGE)
```

**Note:** If the FI parameter resolves to a value that cannot be relocated or pointer value, the ORD parameter must also be a value that cannot be relocated or pointer value.

**ID=recordid**

specifies the record ID used to retrieve the record; for example:

```
ID=AA
```

**Note:** If omitted, no record ID check is done and a warning message is produced.

**ALTID=altidlbl**

specifies the label of a statement that expands to an alternate ID table that is used for recoup chain chase processing. Recoup processing chases the first record ID that matches the information supplied in the table. The statement at the specified label must have the following format:

```
altidlbl INDEX TYP=AIDTBL,(rid,rcc,dscrloc),(rid,rcc,dscrloc),...
```

Where;

*rid* is a 2-character or 4-hexadecimal alternate record ID.

*rcc*

is a 1-character or 2-hexadecimal record code check (RCC) of the alternate record ID.

*dscrloc*

specifies the location of the GROUP macro statement that also describes the data record when retrieved (located in the same descriptor container record). If the record contains no more references to chain chasing (including overflow or forward chaining), *dscrloc* can be omitted, but the preceding comma must still be specified.

## INDEX

Recoup processing compares the record ID and RCC of a record to the information found in the alternate ID table. If the information matches an item, the record is chain chased using the GROUP macro statement at the descriptor location (*dscrloc*). In the following example, assume that recoup processing found the record ID of JJ with an RCC of 01 in the record that was at label JX0FAD. This matches the first ID in the alternate record table; therefore, the record is chain chased using the GROUP macro statement at label JUREC1.

```
INDEX TYP=F,ALTID=TEST,FA=JX0FAD)
TEST INDEX TYP=AIDTBL,(JJ,01,JUREC1),(FC01,1,FCTAG),(BB,,)
JUREC1 GROUP USE=DCSR,
```

**Note:** You cannot specify the ALTID parameter with the ID, RCC, or DSCR parameter.

### **ACODE=***acodeloc*

specifies the label where there is code to run after finding the pool address; for example:

```
ACODE=PD3AR
```

### **BCH=***backchn*

specifies the label of the backward chain address from the last record in the chain; for example:

```
BCH=NC0BCH
```

### **CODE=***codeloc*

specifies the location within the descriptor container record of code to run prior to locating imbedded file addresses; for example:

```
CODE=PREFIND
```

This is useful in cases where certain fields in a data record can be tested to determine if the data record has no imbedded addresses.

### **DSCR=***dscrloc*

specifies the location of the GROUP macro statement that also describes the data record when retrieved (located in the same descriptor container record). The following example shows the DSCR parameter pointing to GROUP3, which is a GROUP macro statement that also describes the data record:

```
INDEX TYP=F, DSCR=GROUP3, ...
```

```
GROUP3 GROUP USE=DCSR, ...
```

### **DUPEELIM**

specifies whether or not to use the special recoup chain-chasing indicator (RCI) algorithm that eliminates duplicate chain chasing.

#### **NO**

does not check pseudo directories to prevent finding a pool that has already been chain chased.

#### **YES**

checks pseudo directories to prevent finding a pool that has already been chain chased.

**Note:** DUPEELIM=YES can only be specified when USE=BASE and GRP=(*grp*id,RCI) are specified in the associated GROUP macro statement, where *grp*id is the group name that is assigned to a specific group of record IDs.

```
INDEX TYP=F,FA=AA0AG1,ID=CC,DUPEELIM=YES
```

```
INDEX TYP=V,ID=CA,FI=I80IFA,LI=L'I80IFA,
      CNT=(N,I80NENT),DSCR=XXXX,DUPEELIM=YES
```

```
INDEX TYP=M,ID=CA,FI=I80IFA,LI=L'I80IFA,
      NAB=(N,I80NENT),DSCR=YYYY,DUPEELIM=YES
```

```
INDEX TYP=C,ID=FC32,DUPEELIM=YES
```

**FA**

specifies the location of the file address, where:

*faloc*

is a displacement to the file address, where *faloc* is one of the following:

- A DSECT tag representing the location of the file address; for example:

```
FA=PD1IFA
```

- An absolute displacement from the start of the first item, expressed as (*alpha,disp*), where *alpha* is any alphabetic character and *disp* is the displacement to the file address; for example:

```
FA=(N,40)
```

*faptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the file address, where *faptr* is one of the following:

- An absolute size and DSECT tag of the location of the file address pointer; for example:

```
FA=(2,VR1FAP)
```

- A calculated size and DSECT tag of the location of the file address pointer; for example:

```
FA=(L'VR1FAP,VR1FAP)
```

- An absolute displacement from the start of the first item, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 16 in the following example:

```
FA=(N,2,16)
```

**Note:** If the FI parameter is specified and resolves to a value that cannot be relocated or pointer value, the FA parameter must also resolve to a value that cannot be relocated or pointer value.

**FVN=versionnbr**

specifies the file version number used by the TPFDF product to identify file structures that have blocks with different layouts than the prime blocks; for example:

```
INDEX TYP=S,FVN=1
```

**MAC=macro\_name**

specifies the name of the data macro describing the data record containing embedded addresses.

**RCC**

specifies the location of the record code check (RCC).

**(P,dtag)**

If the RCC is at a location in the prime record; for example:

```
RCC=(P,NCORCC)
```

## INDEX

### **(I,dtag)**

If the RCC is at a displacement in an item; for example:

```
RCC=(I,TAG1)
```

### **(C,x)**

If the RCC is a constant (any number from 0 to 255); for example:

```
RCC=(C,246)
```

### **REG=reg**

specifies the register to be used as the base for the DSECT specification; for example:

```
REG=R7
```

**Note:** If the REG parameter is specified, the macro generation assumes that the request is to generate the DSECT only.

### **SUFFIX=suffix**

specifies the suffix to be used on the DSECT definition, where *suffix* is an alphabetic character; for example:

```
SUFFIX=Q
```

### **CNT=itemcnt**

specifies the item count, where *itemcnt* is specified as an absolute value or specified as a length and location that contains the item count value.

The following example shows an item count specified as an absolute value:

```
CNT=(N,20)
```

The following example shows an item count specified as a value specified for 2 bytes starting at location PD1TAG:

```
CNT=(2,PD1TAG)
```

### **CREATE**

specifies whether an entry control block (ECB) is to be created with the retrieved record.

#### **YES**

specifies that an ECB is created with the retrieved record, which provides faster processing time and sometimes avoids single threading (usually where there is only one fixed level record with many embedded addresses that also may have many embedded addresses).

If you specify CREATE=YES, more ECBs are created than were specified with the ECB parameter in the primary GROUP macro.

#### **NO**

specifies that an ECB is not created with the retrieved record.

### **FAT**

specifies the type of file address specified with the FA parameter.

#### **FA4**

specifies a 4-byte file address.

#### **FA8**

specifies an 8-byte file address.

#### **PID**

specifies a TPF collection support (TPFCS) persistent identifier (PID).

**FCOD**=*fcodloc*

specifies the location in the descriptor container record of code to be processed for every item in a data record; for example

```
FCOD=ITEMCODE
```

This is used for testing fields in items where the file addresses can be for different record types, or if item is inactive, but file address not cleared.

**Notes:**

1. Register 1 (R1) contains the base of data record.
2. R2 points to item.
3. When TYP=L, R3 has the address of the current ordinal slot, and R2 points to the current item.
4. When TYP=V or TYP=M, R7 has the address of the current subitem, and R2 points to the current item.

**FI** specifies the location of the first item in a record, where:

*filoc*

is a displacement to the first item relative to the start of the record, where *filoc* is one of the following:

- A DSECT tag representing the location of the first item; for example:

```
FI=PD1ITM
```

- An absolute displacement from the start of the record, expressed as (*alpha,disp*), where *alpha* is any alphabetic character and *disp* is the displacement to the first item; for example:

```
FI=(N,40)
```

*fiptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the first item in the record, where *fiptr* is one of the following:

- An absolute size and DSECT tag of the location of the first item pointer; for example:

```
FI=(2,VR1IPT)
```

- A calculated size and DSECT tag of the location of the first item pointer; for example:

```
FI=(L'VR1IPT,VR1IPT)
```

- An absolute displacement from the start of the record, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 16 in the following example:

```
FI=(N,2,16)
```

**Note:** If the FI parameter is specified and resolves to a value that cannot be relocated or pointer value, the FA, LI, ORD, SFA, SI, or SIC parameter must also resolve to a value that cannot be relocated or pointer value.

**FII** specifies the first item index that is used to locate the first valid item in a record, where:

*fiidisp*

is a field that contains the absolute displacement from the start of the record to the first item in a record, expressed as (*alpha, size, namefield*), where *alpha* is any alphabetic character, *size* is the size of the field, and

## INDEX

*namefield* is the name field that contains the displacement. In the following example, the 2-byte field, TAGB, contains the absolute displacement from the start of the record to the first item :

```
FII=(D,2,TAGB)
```

### *fiinbr*

is a field that contains the item number of the first item in a record, expressed as (*size, namefield*), where *size* is the size of the field, and *namefield* is the name of the field that contains the item number of the first item. The first item is item number 0; therefore, the last item number is one less than the number of items. In the following example, the 2-byte field, TAGB, contains the number of the last item in a record:

```
FII=(2,TAGB)
```

**Note:** The FII parameter is used only for fixed-length items.

**LI** specifies the length of an item or the location of a field that contains the length of an item in a record, where:

*li* is the length of an item in a record, where *li* is one of the following:

- An assembler expression that resolves to an absolute value; for example:

```
LI=L'PD1ITM
```

- An absolute value; for example:

```
LI=32
```

### *liptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the length of an item, where *liptr* is one of the following:

- An absolute size and DSECT tag of the location of the item length; for example:

```
LI=(2,VR1IPT)
```

- A calculated size and DSECT tag of the location of the item length; for example:

```
LI=(L'VR1IPT,VR1IPT)
```

- An absolute displacement from the start of the record, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 12 in the following example:

```
LI=(N,2,12)
```

**Note:** If the FI parameter is specified and resolves to a value that cannot be relocated or pointer value, the LI parameter must also resolve to a value that cannot be relocated or pointer value.

**LII** specifies the last item index that is used to locate the last valid item in a record is in a field that contains the absolute displacement to the last item in a record or the location of a field that contains the item number of the last used item in a record, where:

### *liidisp*

is a field that contains the absolute displacement from the start of the record to the last item in a record, expressed as (*alpha,size,namefield*), where *alpha* is any alphabetic character, *size* is the size of the field, and *namefield* is the name field that contains the displacement. In the following example, the 2-byte field, TAGB, contains the absolute displacement from the start of the record to the last item :

LII=(D,2,TAGB)

*liinbr*

is a field that contains the item number of the last item in a record, expressed as (*size,namefield*), where *size* is the size of the field, and *namefield* is the name of the field that contains the item number of the last item. The first item is item number 0; therefore, the last item number is one less than the number of items. In the following example, the 2-byte field, TAGB, contains the number of the last item in a record:

LII=(2,TAGB)

**Note:** The LII parameter is only used for fixed-length items.

**LSI**

specifies the length of a subitem or the location of a field that contains the length of a subitem in a record, where:

*lsi* is the length of a subitem in a record, where *lsi* is one of the following:

- An assembler expression that resolves to an absolute value; for example:

LSI=L'PD1ITM

- An absolute value; for example:

LSI=32

*lsiptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the length of a subitem, where *lsiptr* is one of the following:

- An absolute size and DSECT tag of the location of the subitem length; for example:

LSI=(2,VR1IPT)

- A calculated size and DSECT tag of the location of the subitem length; for example:

LSI=(L'VR1IPT,VR1IPT)

- An absolute displacement from the start of the record, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 12 in the following example:

LSI=(N,2,12)

**Note:** If the SFA parameter is specified and resolves to a value that cannot be relocated or pointer value, the LSI parameter must also resolve to a value that cannot be relocated or pointer value.

**NAB**

specifies the next available byte or the location of a field that contains the next available byte, where:

*nab*

is the next available byte in a record, where *nab* is an absolute value; for example:

NAB=(N,20)

*nabptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the next available byte, where *nabptr* is one of the following:

- An absolute size and DSECT tag of the location of the next available byte; for example:

## INDEX

NAB=(2,VR1NAB)

- A calculated size and DSECT tag of the location of the next available byte; for example:

NAB=(L'VR1NAB,VR1NAB)

### SFA

specifies the location of the subitem file address, where:

#### *sfalloc*

is a displacement to the subitem file address, where *sfalloc* is one of the following:

- A DSECT tag representing the location of the subitem file address; for example:

SFA=TAGA

- An absolute displacement to the subitem file address relative to the start of the first item, expressed as (*alpha,disp*), where *alpha* is any alphabetic character and *disp* is the displacement to the subitem file address; for example:

SFA=(G,40)

#### *sfaptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the subitem file address, where *sfaptr* is one of the following:

- An absolute size and DSECT tag of the location of the subitem file address pointer; for example:

SFA=(2,VR1IPT)

- A calculated size and DSECT tag of the location of the subitem file address pointer; for example:

SFA=(L'VR1IPT,VR1IPT)

- An absolute displacement from the start of the first item, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 16 in the following example:

SFA=(N,2,16)

**Note:** If the FI or SI parameter is specified and resolves to a value that cannot be relocated or pointer value, the SFA parameter must also resolve to a value that cannot be relocated or pointer value.

**SI** specifies the location of the first subitem in an item, where:

#### *siloc*

is a displacement to the first subitem, where *siloc* is one of the following:

- A DSECT tag representing the location of the subitem file address; for example:

SI=TAGA

- An absolute displacement to the first subitem relative to the start of the first item, expressed as (*alpha,disp*), where *alpha* is any alphabetic character and *disp* is the displacement to the first subitem; for example:

SI=(G,40)

#### *siptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the first subitem, where *siptr* is one of the following:

- An absolute size and DSECT tag of the location of the first subitem pointer; for example:  
SI=(2,VR1IPT)
- A calculated size and DSECT tag of the location of the first subitem pointer; for example:  
SI=(L'VR1IPT,VR1IPT)
- An absolute displacement from the start of the first item, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 16 in the following example:  
SI=(N,2,16)

**Note:** If the FI or SFA parameter is specified and resolves to a value that cannot be relocated or pointer value, the SI parameter must also resolve to a value that cannot be relocated or pointer value.

## SIC

specifies the subitem count (that is, the number of subitems), where:

### *sicnt*

is the subitem count, where *sicnt* is an absolute value; for example:

SIC=(N,32)

### *sicntptr*

is the size and location of the 1- to 4-byte field that contains the pointer to the subitem count, where *sicnt* is one of the following:

- An absolute size and DSECT tag of the location of the subitem count; for example:  
LI=(2,VR1SIC)
- A calculated size and DSECT tag of the location of the subitem count; for example:  
LI=(L'VR1SIC,VR1SIC)
- An absolute displacement from the start of the item, expressed as (*alpha,size,disp*), where *alpha* is any alphabetic character, *size* is the size of the field that contains the pointer, and *disp* is the displacement to the pointer field. The absolute displacement to the 2-byte pointer field is 12 in the following example:  
LI=(D,2,12)

### Notes:

1. When SIC is specified, the LSI parameter is required.
2. If the FI parameter is specified and resolves to a value that cannot be relocated or pointer value, the SIC and LSI parameters must also resolve to values that cannot be relocated or pointer values.

## Entry Requirements

- This macro is required for recoup.
- Online use of this macro is for data use only.
- Offline use of this macro is to create BKDI records on the RCP tape.
- Register 9 (R9) must contain the address of the ECB being processed.
- All parameters that apply to the record structure must be specified during offline use of this macro.

## INDEX

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro must be processed on the main I-stream.
- This macro will be used for recoup only.
- For all exits coded it is the users responsibility to establish addressability to the data DSECT concerned.

### Examples

- The following example shows the online use of the INDEX macro.

```
INDEX REG=R7
```

**REG=R7** R7 is the register to be used as the base for the DSECT specification.

- The following example indicates that a pool address is located at some displacement in a data record by the FA parameter.

```
INDEX TYP=F,FA=PD1WLP,DSCR=PWGP,ID=PW,CODE=PWCODE
```

**TYP=F** File address or persistent identifier (PID) at a fixed location.

**FA=PD1WLP** Displacement of 4-byte file address.

**DSCR=PD1WLP**

When the record is retrieved at PD1WLP, this field is a pointer to another GROUP macro that defines embedded addresses in the data record.

**ID=PW** Record ID.

**CODE=PWCODE**

Pointer to code that is run before finding the data record at PD1WLP.

Example:

**TM** PD1ID1-PD1PD(R1),X'01'

**BZ** 0(R6) not an alpha.

**B** 8(R6) yes, find record.

Where:

**0(R6)** Return if conditions are *not* met; the item will be bypassed and the record not retrieved.

**8(R6)** Return if conditions *are* met; the record will be retrieved.

**R1** Base of record containing the pool addresses.

- File addresses at fixed locations.

1. TYP = F

```
INDEX TYP=F,FA=PD1WLP,ID=PW,DSCR=PWGP,CODE=TEST
```

**TYP=F** Branch Vector.

**FA=PD1WLP** Displacement to file address.

**ID=PW** ID of the record found at FA.

**DSCR=PWGP** Displacement to descriptor for record found at FA.

**CODE=TEST** Pointer to executable code.

2. Explanation: CODE=TEST

The CODE parameter is used to determine if a file address is active or present at FA.

Example:

```
TM PD1ID1-PD1PD(R1),X'01'
B 8(R6) FA is present-process
B 0(R6) Not active bypass
```

**Note:** Register 1 (R1) points to the beginning of the data record, and R6 is the return register.

- Data Records with Standard Headers

This covers data records with a forward chain address at location 8 in the data record and a back chain at location 12. The back chain address must be the last data record in the chain when included as the BCH parameter. Nonstandard headers may be described with this statement also.

1. Example:

```
INDEX TYP=C, ID=AT
```

**TYP=C** Standard forward chain chase.

**ID=AT** Record ID.

**Note:** If the forward chain is at a location other than 8, the file address may be described with the FA parameter. However, every chain has to be at the same location described by FA. The same applies to the BCH parameter.

2. Example:

```
INDEX TYP=C, FA=TI1FOR, BCH=TI1BAC, ID=TM
```

**FA=TI1FOR** is the nonstandard first in the chain

**BCH=TI1BAC** is the nonstandard last in the chain

- Standard Forward Chain Chase

```
TYP = C
```

1. INDEX TYP=C, ID=PR

2. INDEX TYP=C, FA=PR0FCH, ID=PR

3. INDEX TYP=C

4. INDEX TYP=C, FA=PR0FCH, BCH=PR0BCH, ID=PR

All parameters other than the TYP=C are optional with the following exceptions:

1. If the forward chain address is at a location other than 8, the FA parameter must specify the nonstandard location.
2. The BCH parameter is used only when the back chain field specifies the last chain. This is used to stop when a closed loop is found.
3. The ID parameter is not used; the ID of the prime record is used to retrieve all subsequent records.

- Groups of Embedded Pool Addresses

## INDEX

The following example describes a group of embedded pool addresses with a starting location, the length of an item to be used to increment from item to item, and a method to calculate the number of items or file addresses.

```
INDEX TYP=V,FI=EG0ADR,LI=L'EG0ADR,CNT=(N,261),
      ID=EO,DSCR=EOID,CODE=NOLAST
```

- TYP=V** V indicates a group of embedded addresses or persistent identifiers (PIDs).
- FI=EG0ADR** Displacement to the first item.
- FA=0** This parameter is not coded because the items are 4 bytes in length. The FA is added to FI to calculate the final displacement to the file address. Therefore the parameter could be coded as FA=0 or it could be left out and defaulted to 0.
- LI=L'EG0ADR** Increment to step to the next file address location.
- CNT=(N,261)** Number of file addresses is fixed at 261. N means the value 261 is used as a counter.
- ID=EO** Record identification.
- DSCR=EOID** Pointer to a descriptor record of the record when retrieved.
- CODE=NOLAST** Pointer to code before indexing through the data record. R1 at this point will have the base address of the data record.

- Groups of Addresses

```
INDEX TYP=V,NAB=(2,PD1NX),FI=PD1ITM,LI=L'PD1ITM,
      FA=PD9ADD,ID=PR,CODE=PDITM,FCOD=TEST,DSCR=PNR
```

- TYP=V** Branch vector.
- NAB=(2,PD1NX)** Displacement to the next available byte.
- FI=PD1ITM** Displacement to the first item.
- LI=L'PD1ITM** Length of one item.
- FA=PD9ADD** Displacement from FI to the file address.
- ID=PR** PR record ID used to retrieve the record at FA.
- CODE=PDITM** Displacement to executable code.
- FCOD=TEST** Displacement to executable code.
- DSCR=PNR** Pointer to the descriptor of the record found at FA.

Explanation:

1. The next available byte is used to calculate the number of items to process.  
NAB-FI = 07A-03E = 2 items active
2. CODE=PD1ITM

**Registers** R1 contains the address of the beginning of the data record  
R6 return register  
0(R6) means to end processing the data record  
8(R6) means to continue processing the data record

The CODE parameter is used primarily for examining a field in the data record to determine whether the data record is empty or inactive.

3. FCOD=TEST

**Registers** R1 contains the address of the beginning of the data record  
 R14 is the address of the current item, which is incremented by the item length to find the address of next item.  
 R6 is the return register.

Example:

```

CLI 0(R2),X'80'      80=item inactive
BO  0(R6)           Bypass this item
B   8(R6)           Process this item
B   16(R6)          Write item to RCP tape
  
```

**Note:** The write item to RCP tape does not FIND the record from file. This is used when you do not want to find the record at this time. For PNRs, this is used with the MET switch and MET TAPE to speed up processing.

- Groups of Addresses in Groups of Items

```

INDEX TYP=V,FI=PG1LST,NAB=(2,PG1DII),LI=L'PG1LST,ID=PD,
      SIC=PG9DCT,SFA=PG9DPI,LSI=L'PG1IT2,DSCR=PNID
  
```

- TYP=V** Branch vector.
- FI=PG1LST** The displacement to the first item.
- NAB=(2,PG1DII)** The displacement to the next available byte.
- LI=L'PG1LST** The length of one item.
- ID=PD** The PD record ID used to retrieve the record.
- SIC=PG9DCT** The displacement (relative to FI) of the count of subitems.
- SFA=PG9DPI** The displacement (relative to FI) of the first file address.
- LSI=L'PG1IT2** The length of the subitem.
- DSCR=PNID** The pointer to the descriptor of the record found at the file address.

Explanation:

1. The first item may be at the end of a data record, which means that the length of the item (L'PG1LST) is decremented from the first item. The first item may be at the beginning of the record, which means that the length of the item is added to the FI value to process from item to item.
2. The CODE and FCOD parameters have the same conventions as when used with the TYP=V group of addresses or persistent identifiers (PIDs).

- Groups of Ordinal Numbers

The following statement is used to describe data records with groups of ordinal numbers. These fixed records then, in turn, have embedded pool addresses.

1. TYP = L

```

INDEX TYP=L,FI=PQ5ITM,LI=L'PQ5SLT,ORD=(17,1,PQ9QGN),
      FCOD=TAGQC,DSCR=QCR,ID=QC,CNT=(1,PQ5CNT)
  
```

**TYP=L** L indicates a group of ordinal numbers.

## INDEX

**FI=PQ5ITML** The first item starts at PQ5ITM (a displacement from the beginning of the data record).

**LI=L'PQ5SLT** The length of the item.

**ORD=(17,1,PQ9QGN)**

17 is the file address compute (FACE) program equate of the record type.

1 is the length of the field containing the ordinal number.

PQ9QGN is the displacement from the item (PQ5ITM) of the ordinal.

### 2. TYP = L

INDEX TYP=L,FI=QC1UIN,LI=L'QC1UIN,ID=QX,ORD=(#QXRNO,2,QC1UX1),  
CNT=(N,29),FCOD=QUE,DSCR=QXR

**TYP=L** Branch vector.

**FI=QC1UIN** The displacement to the first item.

**LI=L'QC1UIN** The length of one item.

**ID=QX** C'QX' ID for retrieval.

**ORD=(#QXRNO,2,QC1UX1)**

**#QXRNO** FACE equate this record type.

**2** The length of the field containing an ordinal number.

**QC1UX1** The displacement (relative to FI) of an ordinal number.

**CNT=(N,29)** The number of items is fixed at 29.

**FCOD=QUE** The pointer to code to test if item is active.

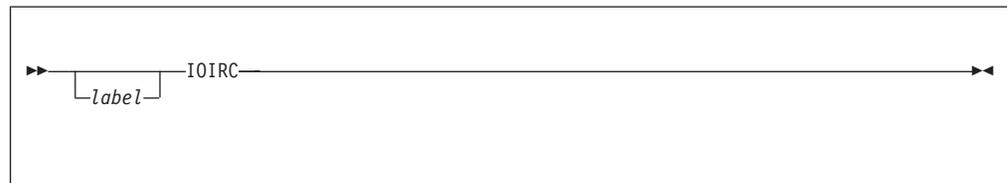
**DSCR=QXR** The pointer to the descriptor of the record retrieved.

Explanation: FCOD=QUE parameter is used to examine some field relative to the beginning of each and every item. (See the previous conventions.)

## IOIRC—Return from CIO Input/Output (I/O) Interrupt Processing

Use this system macro to return control from the device input/output (I/O) interrupt handler to the point of the I/O interrupt.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with the program status word (PSW) key 0.

### Return Conditions

There is no return from the IOIRC macro as such. The macro processes an orderly return to the point of interrupt by reloading the PSW from the low core save area.

### Programming Considerations

This macro can be run on the main I-stream only.

### Examples

None.

---

## IPSDC–Call TCP/IP Native Stack Common Service Routine

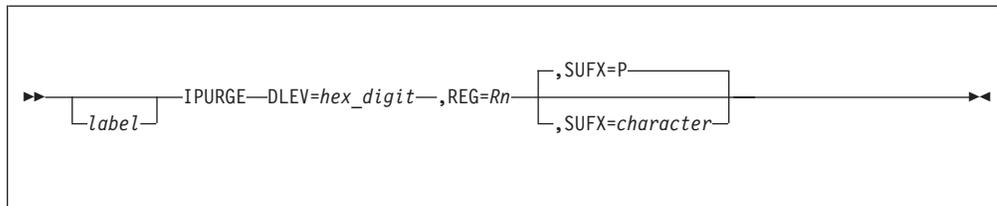
Use this system macro to allow real-time and control program TCP/IP native stack support code to call common TCP/IP service routines that reside in the control program. These routines build and send IP packets, activate and deactivate IP connections, and manage the IP routing table. The interface to IPSDC differs for each service routine.

**Note:** Syntax and parameter descriptions are not provided because this is an object code only (OCO) macro.

## IPURGE—Purge Data from Queue

Use this system macro to purge the presentation services input list queue.

### Format



#### DLEV=*hex\_digit*

The variable *level* indicates a free data level to be used in purging the queue. Specify this as a hexadecimal digit from 0 to F.

#### REG=*Rn*

The variable *Rn* indicates the base register to be used for the Half Session Processing Record (DSECT IHPR). The valid range is R1 through R7, and R14 through R15.

#### SUFX=**P**|*character*

This is an optional parameter that specifies the suffix to be used in the IHPR macro call.

### Entry Requirements

- The calling segment must have set up addressability for the Conversation Control Block (CCB) before invoking this macro.
- The level passed on the DLEV parameter must be free for use.

### Return Conditions

- The level passed on the DLEV parameter is free for use.
- The register passed on the REG parameter is available for use.

### Programming Considerations

- Each invocation of this macro within a segment must have a unique suffix.
- The addressability of the CCB must be set up by the calling segment before invoking this macro.

### Examples

Purge the queue using data level C, specifying the IHPR DSECT in register R5 and associating the letter A as a suffix for the IHPR macro.

```
IPURGE DLEV=C,REG=R5,SUFX=A
```

```
IPURGE DLEV=B,REG=R5
```



## Programming Considerations

- This macro can be run only on an I-stream having affinity with the specified SDA.
- This macro can be used only to interrogate SDAs associated with I/O devices.
- This macro is for use in the control program (CP) only.

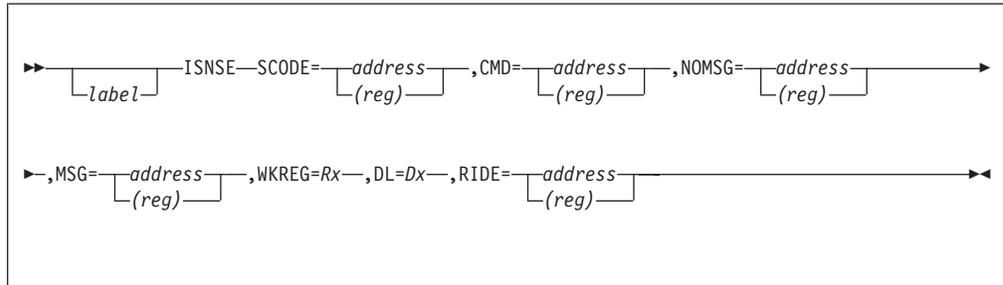
## Examples

None.

## ISNSE—Add an Entry to the Sense Table

Use this system macro to add an entry to the sense (SNS) table that counts the number of resources receiving the same sense code on the same command issued.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **SCODE**

This specifies the sense code that is to be updated in the sense table. The format of the SCODE operand is:

##### *address*

The symbolic address of a two byte field containing the sense code.

##### *(reg)*

A register containing the sense code right justified.

#### **CMD**

This specifies the command that the sense code was received on. The format of the CMD operand is:

##### *address*

The symbolic address of a one byte field containing the command.

##### *(reg)*

A register containing the command right justified.

#### **NOMSG**

This specifies the return address when no message is to be issued. The format of the NOMSG operand is:

##### *address*

The symbolic address of the next instruction to process when no message is to be issued.

##### *(reg)*

A register containing the address of the next instruction to process when no message is to be issued.

#### **MSG**

This specifies the return address when a message is to be issued. The format of the MSG operand is:

##### *address*

The symbolic address of the next instruction to process when a message is to be issued.

*(reg)*

A register containing the address of the next instruction to process when a message is to be issued.

**WKREG=Rx**

This specifies a work register for use by the macro. The contents of the register are unknown upon return. This register must not be R0, R14, or R15.

**DL=Dx**

This specifies a data level that the macro can use to get a RID block. This data level must be free on input and will be returned free on exit from the macro.

**RIDE**

This specifies the RID of the RVT involved. This will be added to the RID block when the sense is already in the sense table. The format of the RID operand is:

*address*

The symbolic address of a two byte field containing the RID.

*(reg)*

A register containing the RID right justified.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The WKREG must not be R0, R14, or R15.
- The DL specified must not already be holding a block.

## Return Conditions

- Control is returned to the MSG label when the sense is not in the table or the table is full. Control is returned to NOMSG label when the sense is already in the table.
- The contents of R14, R15, and the register specified on the WKREG parameter are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro should only be used when processing a negative response.
- This macro issues a CRETC for wait time. System message rates may slow the receipt of the summary message.
- When a register is selected as a parameter it must be in the range R0 through R7 and take the form (Rx).

## Examples

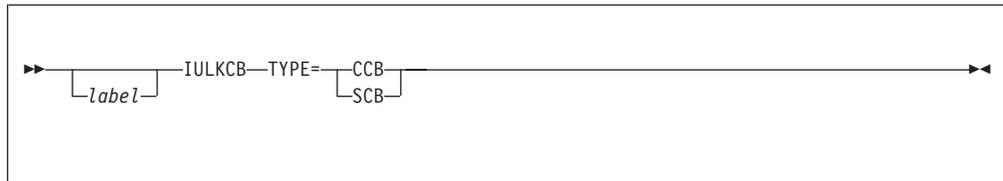
None.

## IULKCB—Unlock a Control Block Area

Use this system macro to unlock a control block area previously locked by the ILCKCB macro.

See “ILCKCB—Lock a Control Block Area Macro” on page 304 for more information about the ILCKCB macro.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **TYPE**

The name of the control block area to be unlocked.

#### **CCB**

The TPF/APPC conversation control block (CCB) is to be unlocked.

#### **SCB**

The TPF/APPC session control block (SCB) is to be unlocked.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The calling segment must have write access (using CINFC W) to the control block to be unlocked.

### Return Conditions

- Control is returned to the next sequential instruction (NSI) after control block is unlocked.
- After the control block is unlocked, the lock fields in the control block contain zeros.
- The contents the R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- Control must not be given up between the ILCKCB and IULKCB macros.
- If the lock is not held by this ECB, the TPF system issues a dump.
- If the lock is not held by any ECB, the TPF system issues a dump.
- This macro is restricted to real-time programs, or CP programs with a valid ECB in R9.

### Examples

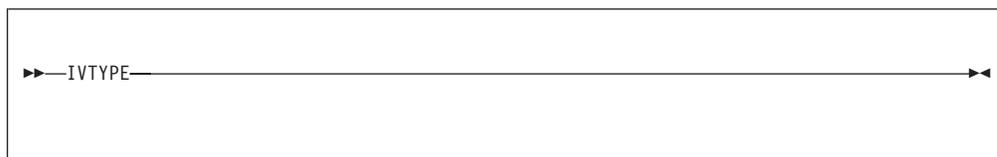
```
MYLABEL IULKCB TYPE=CCB
```

## IVTYPE–GETCC Block Type Verification

Use this system macro to validate the block type parameter for the GETCC macro. This macro contains an AIF statement that you can modify to contain user-defined block types as valid block type equates for the GETCC macro calls.

See *TPF General Macros* for more information about the GETCC macro.

### Format



### Entry Requirements

This macro can only be called from the GETCC macro.

### Return Conditions

Returns to the next sequential instruction (NSI) in the GETCC macro, if the block type is valid, or to the ERR7 label in the GETCC macro call if the block type is not valid.

### Programming Considerations

- This is an assembly time macro.
- You can add the block type equates to the list of types in the macro. This will avoid assembly errors when using non-standard equates in the GETCC macro. The addition is done by modifying the AIF test for legitimate types, for example:

```
AIF ('&P2' NE 'L0' AND '&P2' NE 'L1' AND '&P2' NE 'L2' AND
      '&P2' NE 'L4' AND '&P2' NE '#TPFDBSW' AND
      '&P2' NE '#SW00SRS' AND '&P2' NE '#UI00MS').ERR7
```

The types provided by the TPF system (L0, L1, L2, and L4) must not be changed.

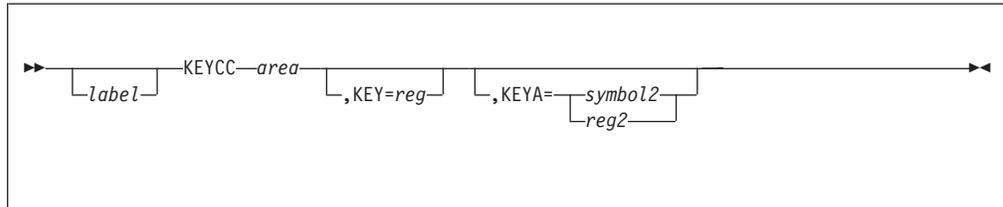
### Examples

None.

## KEYCC—Change Protection Key

Use this system macro to modify a protected main storage area by an application program.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**symbol1**

**GLOBALn**

To unprotect the global area of n

**APLn**

To unprotect the application program area of n in permanent storage.

**KEY=reg**

Optional parameter. If coded, the general register specified will contain the protect key value. Valid registers are R1 through R7, R14, and R15.

**KEYA=symbol2[(reg2)**

Optional parameter. If coded, the address referenced will contain the protect key value. Valid registers are R1 through R7, R14, and R15.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The condition code is saved during processing of this macro.
- On return, the current PSW protection key is identical to the protection key assigned to the specified core area.

### Programming Considerations

- This macro can be run on any I-stream.
- The KEYCC macro requires 4 bytes of storage if the KEY parameter is coded, and 6 bytes of storage if the KEYA parameter is coded.
- The core area is unprotected for the ECB issuing the change protect key (KEYCC) macro.
- After the macro has been issued, no *store* can be processed in any of the previously unprotected areas (ECB, held data blocks, fixed unprotected core) without first issuing the restore protect key (KEYRC) macro.

- Neither the WAITC macro or any macros with implied waits can be issued between the KEYCC and KEYRC macros.
- No fast-link macro can be issued between the KEYCC and KEYRC macros.

## Examples

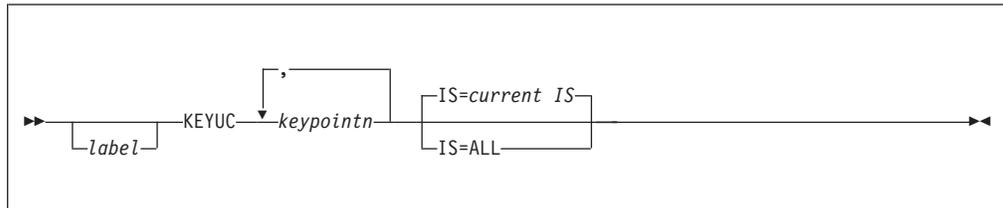
None.

## KEYUC

### KEYUC–Keypoint Update

Use this system macro to request an update of keypoint records through a service call to the control program (CP) KEYUC macro. Up to 8 requests may be entered at any one time.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **keypoint1 through keypoint8**

These are keypointable records as defined in slots 1 through 48 in GLOBA.

**IS ALL** - an optional parameter if the keypoint record is to be updated on all I-streams. The default is the current I-stream only.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The global directory item for each keypoint record referenced with keypoint1....keypoint8, will have an indicator bit set to request filing by the control program (CP).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- KEYUC requires 8 bytes of storage when issued only for the current I-stream. When the IS=ALL option is used, KEYUC will expand to 14 bytes.

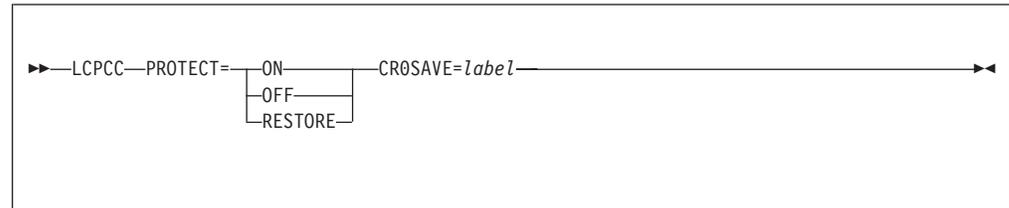
### Examples

None.

## LCPCC–Low Address Protect Set and Restore

Use this system macro to set the low address protect hardware feature. This feature protects the first 512 bytes of storage against alteration by a program regardless of the storage key used by the program.

### Format



#### PROTECT

Specifies if low address protect should be turned on, turned off, or restored to its previous setting.

##### ON

Turns on the low address protect feature.

##### OFF

Turns off the low address protect feature. Control register 0 is saved in the location given by the CR0SAVE parameter.

##### RESTORE

Restores the protect key to that given in the address contained in the CR0SAVE parameter.

It is assumed that a call to LCPCC to turn off the low address protection bit is issued with the same save area as that used by a call to LCPCC to obtain status for the low address protect bit.

It is assumed that an LCPCC call was issued turning off the low address protection bit, using the same save area as the LCPCC call that obtained the low address protect status.

The low address protect bit is retrieved from CR0SAVE and loaded into control register 0. The other bits in the control register are unchanged. If other bits were modified in control register 0 between when LCPCC PROTECT=OFF was issued and when LCPCC PROTECT=RESTORE was issued, these bits are not be affected by this macro.

#### CR0SAVE=label

This label must refer to a 5-byte area beginning on a fullword boundary. It is used to save control register 0 when PROTECT=ON or PROTECT=OFF as well as to restore bit 3 of control register 0 when PROTECT=RESTORE.

### Entry Requirements

E-type programs must be in supervisor state to run this macro. This requires a MONTC call and authorization to use it through correct program allocation.

### Return Conditions

None.

## LCPCCC

### Programming Considerations

- This macro can be coded in the CP or in E-type programs. It uses an inline expansion.
- This macro is restricted to system use only.

### Examples

- This example turns off low address protection and stores control register 0 in the ECB work area.
  - \* `TURN LOW CORE PROTECTION OFF`  
`LCPCCC PROTECT=OFF,CR0SAVE=EBW000`
- This example resumes the protection setting of the control register stored in the ECB work area.
  - \* `RESTORE LOW CORE PROTECTION`  
`LCPCCC PROTECT=RESTORE,CR0SAVE=EBW000`
- This example sets on the low address protection and saves control register 0 in the ECB work area.
  - \* `TURN LOW CORE PROTECTION ON`  
`LCPCCC PROTECT=ON,CR0SAVE=EBW000`

## LEBIC—Load and Shift SS/SSU ID

Use this system macro to do:

- Table Indexing

In a multiple database function (MDBF) environment, several tables are indexed on a subsystem (SS) or subsystem user ID (SSUID) basis through CE1DBI, CE1PBI, CE1SSU, or general registers. If the table element size is a power of 2, you can use this macro to obtain the index value for a specific ID.

- MDBF ID Integrity Checking

This is an option. If CHECK=YES (default) is specified, the specified SS or SSUID is first checked for integrity. If the integrity check fails, (ordinal number plus complement not equal to X'FF'), a system error is issued. For E-type programs, the entry control block (ECB) is exited. The error is exited for C-type programs unless (ENTRY=NO) is specified. If so, a catastrophic system error is issued.

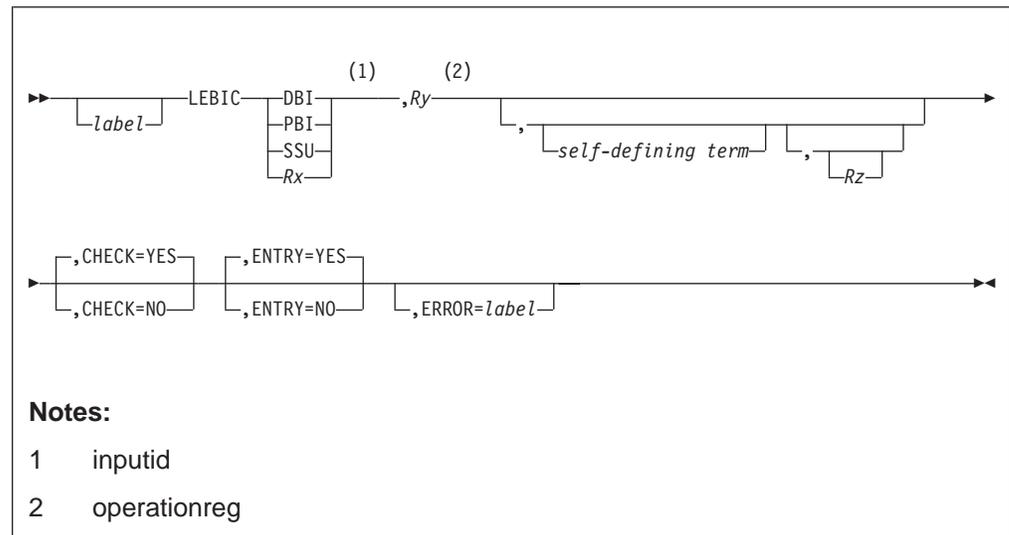
- Conversions of a SS/SSU ID to the SS/SSU Ordinal Number

This is a modification of function 1. If you omit or specify 0 for leftshift, the specified register will contain the ordinal number of the input SS or SSUID.

The reverse case is not supported.

The LEBIC macro expands to inline code for both C-type and E-type programs.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### inputid

Specify one of the following:

#### DBI

If CE1DBI is the source of the input ID.

#### PBI

If CE1PBI is the source of the input ID.

#### SSU

If CE1SSU is the source of the input ID.

**REG**

If the general register specified by REG contains the input ID. See “Programming Considerations” on page 339 for more information. REG must contain a SS or SSU ID in the third and fourth bytes. The first two bytes in the register are irrelevant.

**operationreg**

**REG** or **(,)** See “Programming Considerations” on page 339 for more information. If specified, REG will contain the operation result upon return to the macro caller. If operationreg is not specified and inputid is DBI or SSU, an assembly error is generated. If inputid is a general register and operationreg is not specified, the operation result is returned in the general register specified by inputid.

**leftshift**

**NUM** or **(,)**-where NUM is the number of bits (decimal specification) to be shifted left. If specified, the maximum number of bits to be shifted is 24 when inputid is DBI, PBI, or SSU. If inputid is a general register, then the maximum number of bits to be shifted is 8.

Specifying 0 or **(,)** for leftshift will cause the specified SS/SSU ID to be converted to the SS/SSU ordinal number.

**ecbreg**

**REG** or **(,)**-where REG specifies the base register of the ECB if other than R9 for C-type macro calls when the DBI, PBI, or SSU is specified for inputid. See “Programming Considerations” on page 339 for more information. If **(,)** is specified, R9 is used as the ECB base register when DBI, PBI, or SSU is specified for inputid. For E-type macro calls this positional parameter must be **(,)**.

**CHECK**

Specify one of the following:

**YES**

Specifies integrity checking is required (YES is the default). If the SS or SSU ID given by the macro caller is not valid, the error path is determined by the ENTRY parameter. See the ENTRY parameter for more information.

**NO**

Specifies integrity checking is not required.

**ENTRY**

Is valid only if CHECK=YES and called by C-type programs. If this parameter is coded for an E-type program, an assembly error will occur.

**YES**

Specifies ECB is exited with a system error. YES is the default.

**NO**

Specifies that a catastrophic system error is issued.

**ERROR=label**

Specifies a label to which the control will be passed if an error was detected.

**Entry Requirements**

- CE1PBI if PBI is specified.
- CE1DBI if DBI is specified.
- CE1SSU if SSU is specified.

- A general register if REG is specified for inputid, which must contain a SS/SSU ID in the third and fourth bytes.
- For E-type programs, if inputid is DBI, PBI, or SSU R9 must contain an ECB address.
- For C-type programs when inputid is DBI, PBI, or SSU, if ecbreg is specified, it must specify the general register that contains the ECB address. If ecbreg is not specified, R9 must contain the ECB address.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- The following general registers are valid for the macro parameters:
  - For E-type, use R0 through R7, R14, R15
  - For C-type, use R0-R7, R9(\*), R14, R15
- (\*) R9 is valid only if it does not contain an ECB address.
- E-type programs must have an ECB.
- C-type programs must have an ECB if inputid is not a register.
- The issuing program must not contain the labels CLEBERRE and CLEBPASS.
- Maximum shift is 24 bits if inputid is DBI, PBI, or SSU.
- Maximum shift is 8 bits if inputid is a general register.

## Examples

```
LEBIC SSU,R4,,R9,CHECK=YES
LEBIC SSU,R4,CHECK=NO
```



**READ**

Reads one or more locks in a CF.

**READUSER**

Reads all locks that are held by a particular user on a CF.

**VERIFYCF**

Verifies the connectivity between a processor and a CF.

**ERROR=*label2***

This label specifies the symbolic name of an error routine in which to branch if any error indicators were set.

## Entry Requirements

See the ICFLP DSECT to determine the required input fields and the information included on return in the ICFLP\_DATA control area.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of register 14 (R14) and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The reply code for individual requests must be checked to determine the success or failure of that specific request.
- The following response codes are returned to the caller in the ICFLP\_RSC field.. Use this information to interpret those response codes.

**COMPLETED\_OK**

The requested operation was completed successfully.

**TIMEOUT**

The requested operation was not able to be completed because a timeout occurred.

Submit the operation again using the CF locking restart token provided in the ICFLP\_RT field.

**INV\_RESTOK**

A program error occurred because a CF locking restart token that is not valid was provided for an operation that was resubmitted.

Do the following:

1. Correct the CF locking restart token.
2. Enter the LEMIC macro again.

**DB\_FILLED**

The data block provided with either a READ or READUSER operation is filled and there are more locks to read.

Submit the READ or READUSER operation again using the CF locking restart token provided in the ICFLP\_RT field, as well as a data block that no longer contains critical data.

**LN\_MM**

A list number mismatch occurred.

Do the following:

1. Submit a new operation (such as LOCATELOCK, LOCATEMOD, READ, or READUSER) that will return a valid list number.
2. Submit the original operation again using a valid list number.

**LN\_DNX**

The list number does not exist.

Do the following:

## LEMIC

1. Submit a new operation (such as LOCATELOCK, LOCATEMOD, READ, or READUSER) that will return a valid list number.
  2. Submit the original operation again using a valid list number.
- INS\_SPACE** An error occurred during a read operation because there is not enough space in the data block.
- GLBL\_MM** A global lock manager mismatch occurred.  
Exit the application. If necessary, see your IBM service representative for more information.
- LCL\_MM** A lock manager mismatch occurred.  
Exit the application. If necessary, see your IBM service representative for more information.
- LAU\_MM** A list authority mismatch error occurred.  
Exit the application. If necessary, see your IBM service representative for more information.
- INS\_MB\_SPACE**  
A program error occurred because there is not enough message buffer space available to read the locks in a CF or to read all locks held by a particular user on a CF.  
Exit the application. If necessary, see your IBM service representative for more information.
- ABNORMAL** A program error occurred because of an abnormal locking condition. Some of the set, release, or withdraw commands were not processed by a CF.  
Exit the application. If necessary, see your IBM service representative for more information.
- RRC\_ERROR** An error occurred while processing a MONITOR operation.  
Exit the application. If necessary, see your IBM service representative for more information.
- CVCF\_ERROR**  
The CCCFLC service routines were unable to determine the CF or the list number from the information provided with the operation.  
Exit the application and review the application code to ensure the operation was set up correctly. If necessary, see your IBM service representative for more information.
- CMD\_SUPPR** A program error occurred because the function call was suppressed.  
Exit the application and review the console logs to determine if the CF specified in the operation was correctly added to the CF locking configuration. If necessary, see your IBM service representative for more information.
- CMD\_TERM** A program error occurred because the function call ended.

## Programming Considerations

- This macro can be run on any I-stream.

- **Attention:** Using this macro can cause corruption of the locking control information on the CF. This corruption can lead to errors that cannot be predicted and online database integrity problems.
- A WAITC macro is issued implicitly with this macro.
- The calling program returns storage (specified on the BLOCK parameter) when it is no longer needed.
- If the specified CF or CF locking structure is not valid, a system error is issued.

## Examples

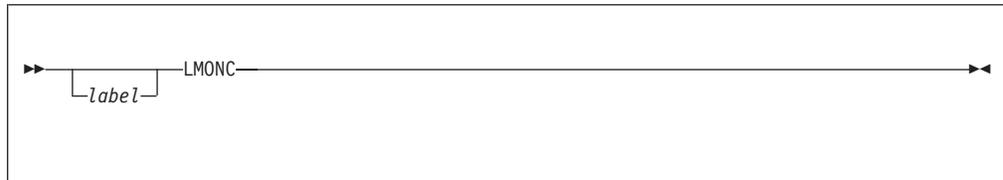
None.

## LMONC—Reset Supervisor State (Problem State)

Use this system macro to change the operating state of the central processing unit (CPU) from supervisor state to problem state. The normal use for this macro is to reset the CPU state following a MONTC macro.

See “MONTC—Set Supervisor State (Monitor Mode)” on page 348 for more information about the MONTC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of registers R0 through R7 are preserved across this macro call. If R10 or R13 are needed by the calling program, they must be saved and restored across the macro call.
- The CPU is in problem state (PSW bit 15 = 1).
- The storage protect key is one (PSW bits 8 - 11 = 1).

### Programming Considerations

- This macro can be run on any I-stream.
- This macro is for system programming use only.

### Examples

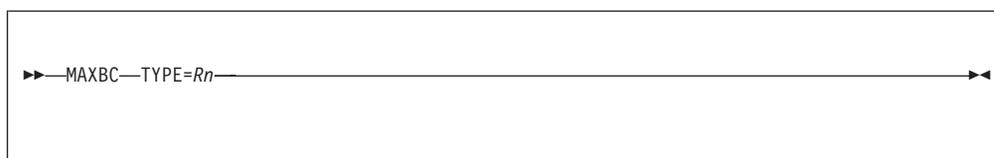
None.

## MAXBC—Get Maximum Number of Storage Blocks

Use this system macro to return the total number of certain storage blocks allocated by the TPF system. The size of the physical data table and the equate for the block are used to determine the size. The allocations of traditional TPF blocks (L0 or 128, L1 or 381, L2 or 1055, L4 or 4 K) can not be queried because these block are not implemented as independent block types. The block types supported are LIOCB, LECB (L3), LSWB, LCOMMON, and LFRAME.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### TYPE=*Rn*

The register that contains the physical storage block type equate. R0 is not valid for all calls and R8 through R13 are not valid for all E-type calls.

Valid types, as defined in CLHEQ, are:

#### LIOCB

I/O control block

#### LECB

Entry control block (ECB)

#### LSWB

System work block (SWB)

#### LCOMMON

Shared working storage block

#### LFRAME

ECB unique working storage block

The MAXBC macro does not support logical blocks carved from other blocks. The block type values for MAXBC that are not valid follow here.

**L0** 128 Byte Block

**L1** 381 Byte Block

**L2** 1055 Byte Block

**L4** 4 K Block

### Entry Requirements

- The register specified on the TYPE parameter must contain the value for the block type desired.
- R0 cannot be specified.
- For E-type programs R10 must be available as a scratch register.

## MAXBC

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register specified on the TYPE parameter contains the number of blocks allocated for the specified block type. The contents of all other registers are preserved across this macro call.
- The condition code is not saved across the macro call.
- If the block type requested is determined to be in error at run time, the 00074D system error is issued and the ECB is exited.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro may be called by E-type programs or from control program (CP) code.
- R0 cannot be specified for the TYPE parameter.
- All logical blocks are carved from 4 K blocks. Because of this, it is impossible to specify the number of available storage blocks, except 4 K blocks, at any given point.
- The usage of the MAXBC macro requires authorization to issue a restricted macro (CHECK=RESTR) by the \$CKMAC macro.
- A system error dump can occur when servicing a MAXBC request. See *Messages (System Error and Offline)* for more information.

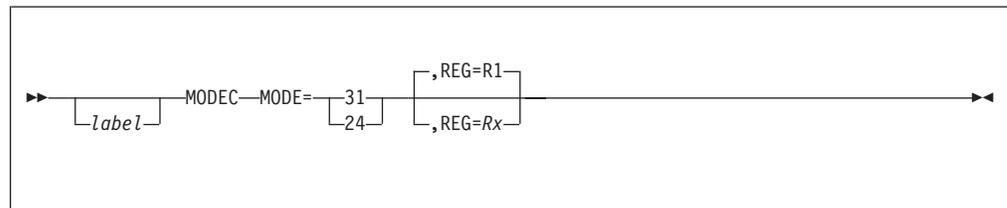
### Examples

None.

## MODEC–Change Addressing Mode

Use this system macro to set the TPF system to 24-bit addressing mode or 31-bit addressing mode.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### MODE

Specify one of the following:

**31** This argument indicates that 31-bit addressing mode is to be set.

**24** This argument indicates that 24-bit addressing mode is to be set.

### REG=Rx

This parameter is optional. It provides a work register for the macro to use.

When coded in a real-time program, valid registers are: R1 through R7, R14, and R15. When coded in any other type of program, any register except R0 can be used. If the REG parameter is not coded the register defaults to R1.

## Entry Requirements

All entry requirements are fulfilled by the proper use of the parameters as defined in “Format”.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Upon return, the requested addressing mode is set in the PSW.
- The work register specified by the REG parameter is not saved. The contents of the remaining registers are unchanged.
- The condition code is only changed when the parameter MODE=31 is used. The condition code remains unchanged when the parameter MODE=24 is used.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro is intended for system use, not for general purpose use.
- The contents of the work register specified by the REG parameter are not saved.
- MODEC cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES).

## Examples

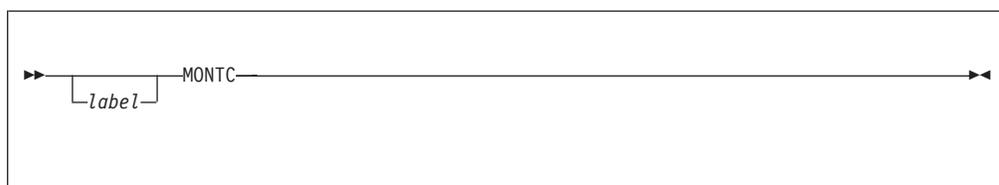
None.

## MONTC–Set Supervisor State (Monitor Mode)

Use this system macro to change the operating state of the central processing unit (CPU) from problem state to supervisor state. Supervisor state allows processing of privileged instructions such as set system mask (SSM) and start input/output (SIO). Use of this macro is restricted to utility type programs (such as disk copy and tape switch) that have special requirements for this state (system programming use only).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X	X	

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown.
- The contents of registers R0 through R7 are preserved across this macro call.
- The CPU is in supervisor state (PSW bit 15 = 0).
- The storage protect key is zero (PSW bits 8 through 11).

### Programming Considerations

- This macro can be run on any I-stream.
- Care should be exercised when operating in this state. Any area above the 512-byte line can be modified in low main storage with just a MONTC. If low main storage below the 512-byte line is to be modified, the LCPCC macro with low memory protection disabled must be issued first. Afterward LCPCC must be issued to re-enable low memory protection.
- A program should operate in this state for the shortest possible time and then issue a LMONC macro to return to problem state.
- Supervisor state is maintained across all other macros, which can be issued by the program.
- Registers R10 and R13 must be saved and restored if required by the calling program.

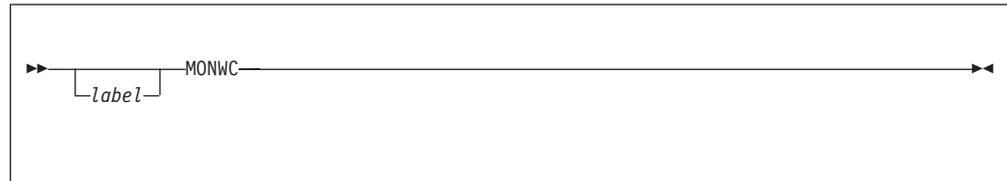
### Examples

None.

## MONWC–Suspend ECB, Pending I/O Completion

Use this system macro to enable the WAITC service routine to suspend the entry control block (ECB) until all pending input/output (I/O) for the ECB completes. This macro is processed **only** by the WAITC service routine.

### Format



*label*

A symbolic name may be assigned to the macro statement.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- This macro is called by the WAITC service code only when pending I/O exists for an ECB.

### Return Conditions

- Control is returned to the next sequential instruction (NSI) in the WAITC service routine.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro is for use in the control program (CP) only, and is used by the WAITC macro. See *TPF General Macros* for more information about the WAITC macro.
- This macro can be run on any I-stream.
- This macro is for use only by the WAITC service routine.
- The service routine will save the current SVC\_OLD\_PSW and the calling program's return address in ECB field CE1PSW prior to transferring control to the system task dispatcher (CPU loop).
- The calling program regains control at the location saved in field CE1PSW when all pending I/O completes.

### Examples

None.

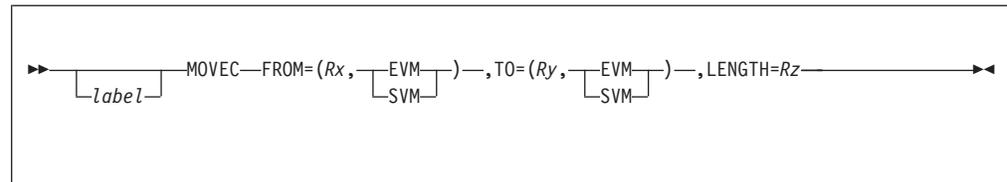
## MOVEC

### MOVEC—Move Data Between EVM and SVM

Use this system macro to move data between an entry control block (ECB) virtual memory (EVM) address space and a system virtual memory (SVM) address space. This macro allows an ECB-controlled program to read and modify storage that is not part of its own address space.

Required Authorizations			
Key0	Restricted	System	Common Storage
X			

### Format



*label*

A symbolic name may be assigned to the macro statement.

#### FROM

This parameter specifies the location of the data from which data will be moved. You must specify both a register, and EVM or SVM.

*Rx* Specifies a general purpose register from R0 through R7. It contains the 31-bit address from which data will be moved.

#### EVM

Specifies that the address is in the ECB virtual memory (EVM).

#### SVM

Specifies that the address is in the system virtual memory (SVM).

#### TO

This parameter specifies the location where the data specified by the FROM parameter will be placed. You must specify both a register, and EVM or SVM.

*Ry* Specifies a general purpose register from R0 through R7. It contains the 31-bit address to which data will be moved.

#### EVM

Specifies that the address is in the ECB virtual memory (EVM).

#### SVM

Specifies that the address is in the system virtual memory (SVM).

#### LENGTH=Rz

This parameter specifies a general purpose register from R0 through R7 that contains the number of bytes of data that will be moved from the FROM location to the TO location.

### Entry Requirements

- Any program issuing the MOVEC macro must be allocated with authorization to process with protect key zero (CINFC WRITE capability).
- This macro must be called from an ECB-controlled program.

- The value you specify for the FROM and TO parameters must be a valid data address; that is, the high order bit must not be turned on.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The data is moved from the FROM location to the TO location.

## Programming Considerations

- The usage of the MOVEC macro requires key 0 write authorization (CHECK=KEY0) and authorization to issue a restricted macro (CHECK=RESTRICT) by the \$CKMAC macro.
- This macro may be processed on any I-stream.
- The following combinations are valid:
  - From EVM to SVM
  - From SVM to EVM
  - From EVM to EVM
  - From SVM to SVM.

**Note:** The EVM-EVM combination is restricted to movement within ECB virtual memory (EVM), not between EVMs.

- System error dumps can occur when servicing a MOVEC request. See *Messages (System Error and Offline)* for more information about system errors.

## Examples

This code shows the FROM address being taken from a DSECT field, the TO address being taken from a core block reference work, and the length being calculated as the difference between two addresses. The next address in the chain is loaded and the data moved.

```

LA    R6,LK40BF          TOP OF LINK QUEUE ADDRESS (LK4KC)
L     R3,CE1CR0         A(block) to use for MOVEC (EB0EB)
LA    R7,CM8XTA-CM8CHNA Need only a few bytes (CM8CM)
ICM  R6,B'1111',0(R6)  LOAD BASE WITH NEXT IN CHAIN
BZ   CMMMDONE          We've gone through queue
SPACE
CMMTEST DS  0H
MOVEC FROM=(R6,SVM),TO=(R3,EVM),LENGTH=R7

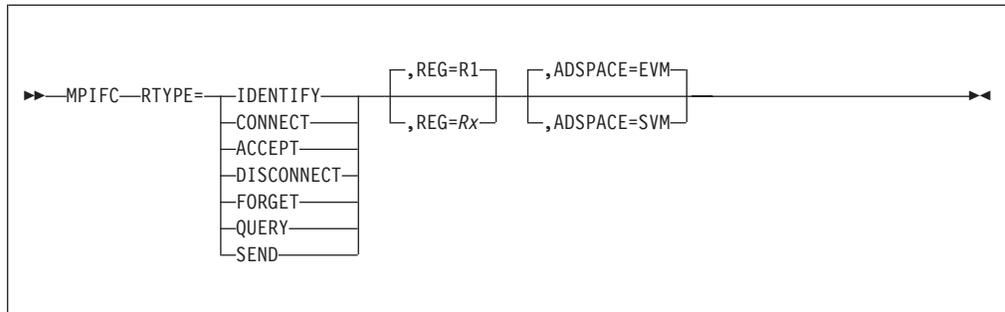
```

## MPIFC–Request MPIF Service

Use this system macro to request Multi-Processor Interconnect Facility (MPIF) services. See “Format” for more information about the valid request types.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### RTYPE

This required parameter specifies the type of request being made by the user. The valid options are:

#### IDENTIFY

This function establishes the existence of a unique, named user that will communicate with other users of MPIF.

RTYPE=IDENTIFY means that REG contains a pointer to a parameter list containing the user identification. If REG is not specified, R1 is assumed. The parameter list is in the form of the DCTMUP DSECT.

An IDTOK is assigned using the IDENTIFY function. It associates the user with MPIF resources and is required for other MPIF functions. The token is returned in the IDTOK field in the user’s parameter list.

#### CONNECT

This function attempts to establish a logical connection between the calling user and the user named in the parameter list.

#### ACCEPT

This function attempts to complete a logical connection between the calling user and the user name in the parameter list.

#### DISCONNECT

This function breaks a logical connection between the calling user and the user named in the parameter list.

#### FORGET

This function is the opposite of IDENTIFY. Any connections involving the calling user are disconnected. All knowledge of the user is eliminated.

#### QUERY

This function formats directory information into an area of storage provided by you.

**SEND**

This function normally schedules the transfer of data across a logical connection. SEND is also used to control pacing values to the other side of a connection.

**REG=R1|Rx**

The symbolic name of a general register (R0 through R7 inclusive) which contains the address of the parameter list, defined in DCTMUP, associated with the service request. R1 is the default.

**ADSPACE**

This parameter is only valid when issued by C-type code and is used to indicate the address space in which the code is processing.

**EVM**

The program that issued the MPIFC is running in the ECB virtual memory (EVM). This is the default.

**SVM**

The program that issued the MPIFC is running in the system virtual memory (SVM).

## Entry Requirements

R14 and R15 must be available for use by this macro.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- IDENTIFY:
  - R15 contains a return code describing the completion of the IDENTIFY processing using the following indications.

**Return Code    Meaning**

00	Normal completion. The MPIF IDENTIFY token (IDTOK) is returned in field IDTOK of the user's parameter list.
04	The user name (MYNAME of the IDENTIFY parameter list) is in use.
12	Unable to process due to lack of MPIF control storage.
20	At least one parameter value is not valid.

- CONNECT:
  - R15 contains a return code describing the completion of the CONNECT processing using the following indications.

**Return Code    Meaning**

00	Normal completion; request proceeding.
04	Connections between 2 users exist over all paths (used when multipathing is selected).
08	No active paths exist to the specified user.
208	At least one parameter value is not valid.

- ACCEPT:
  - R15 contains a return code describing the completion of the ACCEPT processing using the following indications.

**Return Code    Meaning**

## MPIFC

- 00 Normal completion; request proceeding.
- 20 At least one parameter value is not valid.
- DISCONNECT:
  - R15 contains a return code describing the completion of the DISCONNECT processing using the following indications.

<b>Return Code</b>	<b>Meaning</b>
00	Normal completion; request proceeding.
00	At least one parameter value is not valid.
- FORGET:
  - R15 contains a return code describing the completion of the FORGET processing using the following indications.

<b>Return Code</b>	<b>Meaning</b>
00	Normal completion; request proceeding. No further reference to the entity is possible.
20	20–At least one parameter value is not valid.
- QUERY:
  - R15 contains a return code describing the completion of the QUERY processing using the following indications.

<b>Return Code</b>	<b>Meaning</b>
00	Normal completion (for example, a successful QUERY).
04	Feedback area too small, contains partial response.
20	At least one parameter value is not valid or a format problem with the parameter list.

    - The address of the QUERY reply parameter area will be returned in the user specified register (REG parameter).
- SEND:
  - R15 contains a return code describing the completion of the SEND processing using the following indications.

<b>Return Code</b>	<b>Meaning</b>
00	Normal completion; the SEND is scheduled. The SEND parameter list contains the connection sequence number assigned to the SEND. The initial SEND will be assigned a sequence number of 1. The sequence number is a 32 bit logical value and will wrap to zero.
04	MPIF output queue has exceeded the maximum depth. Retry this SEND request again shortly.
08	The path or connection is not active.
12	Currently unused. Reserved for IBM use only.
16	No pacing credit available. The buffer has not been sent. It is returned to the caller.
20	At least one parameter value is not valid.
- The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro may be processed on any I-stream.
- The MPIFC macro has an interface that is not guaranteed across releases of the TPF system. Unauthorized use of this macro may expose you to interface or processing errors.
- This macro is intended for use by system programmers doing system level coding for support or utility programs.
- You cannot include the parameter list in the same core block with data area 2.

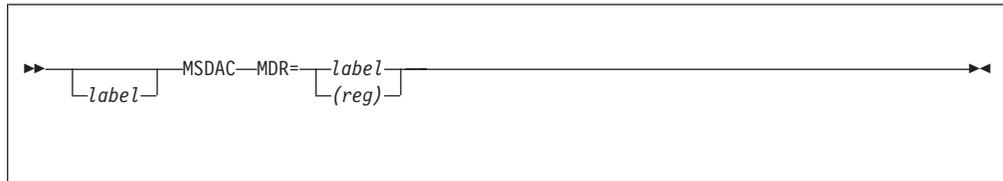
## Examples

None.

## MSDAC—Mount a Symbolic Device Address (SDA)

Use this system macro to perform a software mount of a symbolic device address (SDA) that is associated with an input/output (I/O) device. This allows the SDA to be used by the TPF system to perform I/O operations through common I/O (CIO).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **MDR**

The mount device request (MDR) block, which can be either:

#### *label*

A symbol can be assigned to an area of memory that contains an MDR as defined by the DCTMDR DSECT.

#### *(reg)*

A register that contains the address of the MDR.

### Entry Requirements

- The MDR parameter must be coded with the SDA address and parameters required to do the mount.
- The overlay option may be specified to switch the SDA parameters by setting the MDROVL bit in field MDRFG1 of the DCTMDR mount device request block DSECT. The code overlays the old parameters with the new parameters, except for the I/O interruption subclass, which cannot be changed. With overlay the code does not replace individual parameters but does a complete replace. Those parameters that are not to be replaced must still be specified in the MDR.

**Note:** If the SDA is not already mounted then the overlay option is ignored and the mount is treated as a normal mount request.

- The caller may request that program-controlled interrupts (PCIs) for this device are to be treated as normal interrupts by setting the MDRNOPCI bit in field MDRFG1 of the DCTMDR mount device request block DSECT. This causes PCIs that are received while PIO is active to be stacked and presented after PIO completes, rather than presented immediately. This option may be requested for devices that can receive PCIs but whose interrupt handlers are not coded to handle interrupts presented during system error processing.
- I/O interrupts must be masked before macro invocation.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- R15 will contain one of the following return codes:

Return Code	Meaning
-------------	---------

0	The SDA mount was successful.
---	-------------------------------

- 4                    The SDA is in use.
- 8                    The SDA is not valid.
- R0–R14 are unchanged.
- If the mount is successful, the channel path identifier (CHPID) type and the subchannel path-available mask are returned to you through the MDR block.

## Programming Considerations

- This macro can be run only on an I-stream having affinity with the specified SDA.
- This macro can be run only to mount SDAs that are associated with I/O devices.

## Examples

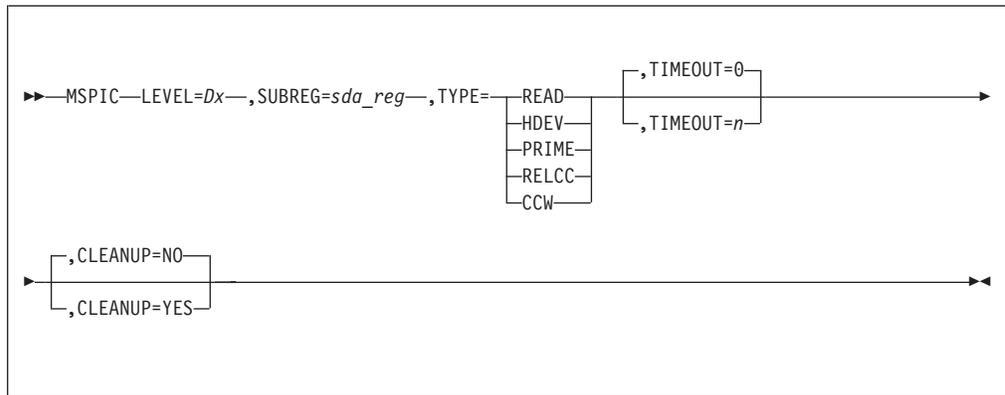
None.

## MSPIC–Control MPIF Device

Use this system macro with the Multi-Processor Interconnect Facility (MPIF) support programs to schedule device control functions other than the normal read or write channel programs generated by MPIF itself.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### LEVEL=*Dx*

A core block reference word and associated file address reference word (D0-DF) must be specified.

### SUBREG=*sda\_reg*

This required parameter specifies the general register (R0 through R6) which contains the relevant symbolic device address (SDA).

### TYPE

This required parameter specifies one of the following options of the MSPIC macro:

#### READ

Read Buffer.

#### HDEV

Deactivate the channel program for the specified SDA pair.

#### PRIME

Prime the specified SDA.

#### RELCC

Release the multiple system request block (MSRB).

#### CCW

Process the user-defined channel program (CCW). CCWs must be Format 1.

**Note:** This option requires processing of a WAITC macro to verify the results, and is therefore restricted to E-type programs.

**TIMEOUT**

Timeout value for SENSE ID channel program. 0 is the default value. This parameter must be 0 unless the TYPE parameter is coded as CCW.

**CLEANUP**

Specify one of the following:

**YES**

Clean-up all resources associated with a path.

**NO**

No cleanup requested.

NO is the default option. YES is only valid when the TYPE parameter is coded as HDEV.

**Entry Requirements**

- For the RELCC option, the core block reference word of the specified data level must contain the address of the MSRB to be released.
- For the CCW option, the core block reference word of the specified data level must contain the address of the channel program (that is, CCWs) to be processed.

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- Following the CCW option the contents of the core block reference word at the specified data level is unchanged.
- R15 contains a return code as follows.

**Return Code    Meaning**

00                    Successful processing.

00                    A value in the SUBREG register is not valid.

08                    Subchannel not attached.

- Following a WAITC, for the CCW option, the following parameters will be returned on I/O error conditions:
  - The error reporting fields in the ECB (CE1SUD, CE1SUG, or CE1SUC) identify the I/O error.
  - The channel status word (CSW) will be contained in the second ECB work area starting at location CE1WKB.
  - The sense data will be contained in the second ECB work area starting at location CE1WKB+8. 32 bytes are reserved for this sense data field.
- The contents of all other registers are preserved across this macro call.

**Programming Considerations**

- This macro can only be run on the I-stream where MPIF is active.
- The CCW option requires that a WAITC macro be issued following the MSPIC macro processing. This restricts the use of the CCW option to E-type programs only.

**Examples**

None.

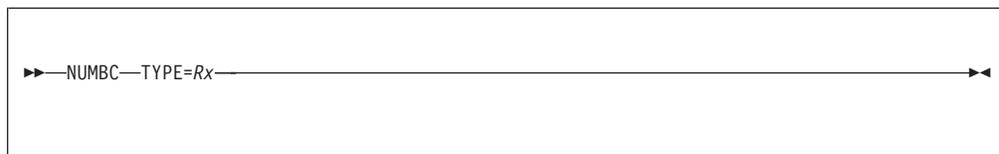
## NUMBC

### NUMBC—Query Number of Storage Blocks Available

Use this system macro to return the number of blocks available for a specified physical block type.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### TYPE=Rx

The specified register must contain the physical storage block type value. Valid physical storage block types are defined in CLHEQ. L0, L1, L2, and L4 are no longer valid. R0 is not valid for all calls and R8 through R13 is not valid for all E-type calls.

### Entry Requirements

- The register specified on the TYPE parameter must contain a valid storage block type.
- In E-type programs R10 must be available as a scratch register.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of the register specified by TYPE contains the number of available storage blocks for this block type.
- The contents of register R10 is unpredictable.
- The condition code is not saved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- R0 and R13 cannot be specified for TYPE.
- The usage of the NUMBC macro requires authorization to issue a restricted macro (CHECK=RESTR) by the \$CKMAC macro.
- A system error dump can occur when servicing a NUMBC request. See *Messages (System Error and Offline)* for more information.

### Examples

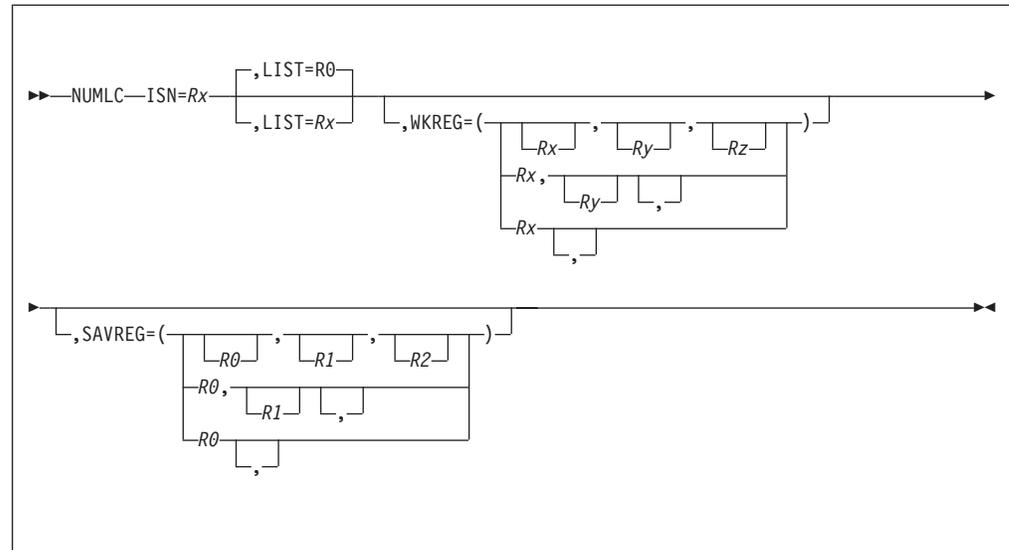
None.

## NUMLC—Get Count of Blocks Queued on a Dispatch List

Use this system macro to return to the caller the number of blocks currently queued on the specified system task dispatcher list.

Required Authorizations			
Key0	Restricted	System	Common Storage
X	X		

### Format



#### ISN=R<sub>x</sub>

The register specified on this input parameter contains the I-stream number for the CPU whose system task dispatcher list is to be queried. If this parameter is omitted, the I-stream that the request is issued on will be queried.

#### LIST=R<sub>0</sub>|R<sub>x</sub>

The register specified on this input parameter contains a list equate value. Equate values are assigned to the cross, ready, input, and defer lists. If the parameter is left out, the default assignment is R0. Valid equate values are:

- #CLHCRS**  
Cross List
- #CLHRDY**  
Ready List
- #CLHINP**  
Input List
- #CLHDEF**  
Defer List

#### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you

## NUMLC

expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

If the macro is coded in a real-time segment, R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register used on the LIST parameter will contain the number of blocks found on the specified system task dispatcher list. When invoked from a real-time segment, all other user registers are preserved across this macro call. When invoked from the control program (CP), the registers specified on the WKREG parameter will be used to save the contents of any volatile registers specified on the SAVREG parameter. All other registers will be returned intact.
- The condition code is not saved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- If LOCK=YES and the specified program is file resident, then the program remains in storage until the RELPC macro is issued.
- NUMLC can be used either by realtime programs or by the control program. It is a restricted use macro and requires KEY0 authorization when used by a realtime program.
- The WKREG and SAVREG parameters are used only on macro invocations from the control program (CP). These parameters have no effect on the macro expansions in real-time programs.

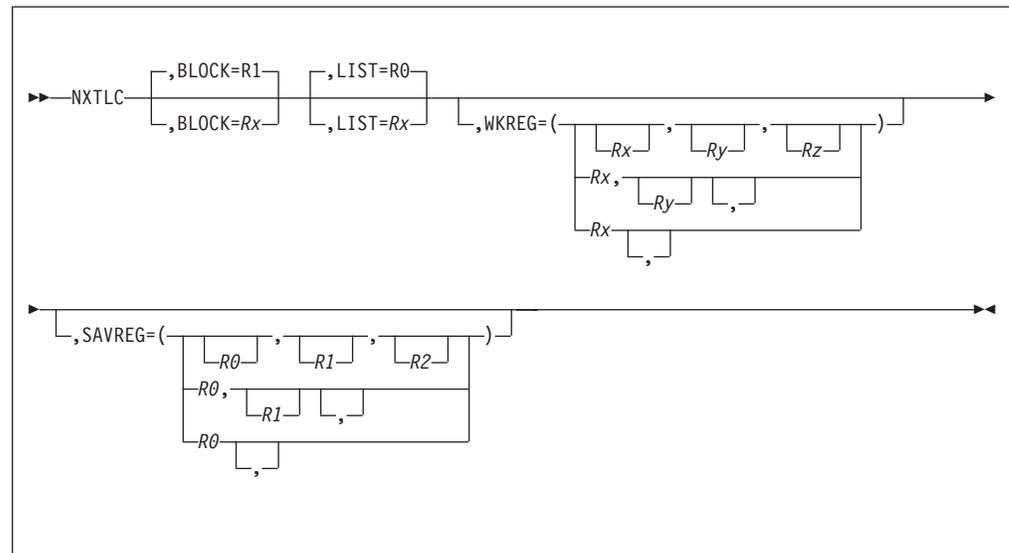
## Examples

None.

## NXTLC—Get Address of Next Block Queued on a Dispatch List

Use this system macro to return to the caller the address of the next storage block queued on the specified system task dispatcher list.

### Format



#### BLOCK=R1/Rx

On input, the register specified contains a value indicating whether to get the address of the first block or the next block on the specified system task dispatcher list. The value 0 means get the address of the first block. Any other value means get the address of the next block on the list following the address specified (as the value).

On output, the register specified contains either the same value or the address of the requested block (for example, the first one or the next one) on the specified system task dispatcher list. The value is unchanged when one of the following occurs:

- The address of the first block was requested but the specified list is empty
- The address of the next block was requested but there are no more blocks on the specified list
- The address of the next block was requested but the specified block is not on the specified list.

The default is R1.

#### LIST=R0/Rx

The register specified on this input parameter contains a list equate value as defined in macro CLHEQ. Equate values are assigned to the Cross, Ready, Input, and Defer lists. If the parameter is omitted, the default assignment is R0. Valid equate values are:

##### #CLHCRS

Cross List

##### #CLHRDY

Ready List

## NXTLC

### #CLHINP

Input List

### #CLHDEF

Defer List

### SAVREG

The specified *volatile* registers will be saved by the macro in the stack area or in the registers specified by the WKREG parameter. Up to 3 registers can be specified. Those registers are R0, R1, R2. If this keyword is omitted, none of the 3 registers will be saved. You should not save a volatile register if you expect it to contain an output parameter. The register will be overwritten with its original contents, since the reload is the last thing performed by the macro.

### WKREG

The specified symbolic register names are free to be used by the macro to save the *volatile* registers coded on the SAVREG parameter. Up to 3 registers can be specified, but the standard linkage registers R13 through R2 cannot be used here. This parameter is used in conjunction with the SAVREG parameter to generate efficient code and enhance the performance of the macro. The number of registers specified by WKREG should be less than or equal to the number of registers specified by SAVREG. *If this parameter is omitted or not used to its maximum capacity, code optimization is sacrificed.*

## Entry Requirements

The register specified on the LIST parameter must contain a value that corresponds to one of the system task dispatcher lists.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The condition code is set to 0, 1, or 2 as follows.

Condition Code	Meaning
0	The requested block was found successfully. Its address is in the register specified on the BLOCK parameter. It is the last block on the specified list.
1	The requested block was found successfully. Its address is in the register specified on the BLOCK parameter. It is not the last block on the specified list.
2	The specified list is empty. The specified block (for example, the block whose address is specified on the BLOCK parameter) is not on the specified list.

- The register specified on the BLOCK parameter contains either the same value (in the condition code 2 case) or the address of the requested block (for example, the first one or the next one) on the specified system task dispatcher list (in the condition code 0 or 1 cases). The value is unchanged when the address of the:
  - First block was requested but the specified list is empty
  - Next block was requested but there are no more blocks on the specified list

- Next block was requested but the specified block is not on the specified list.
- The registers specified on the WKREG parameter will be used to save the contents of any volatile registers specified on the SAVREG parameter. All other registers will be returned intact.

## **Programming Considerations**

- This macro can be run on any I-stream.
- This macro is for use in the control program (CP) only.

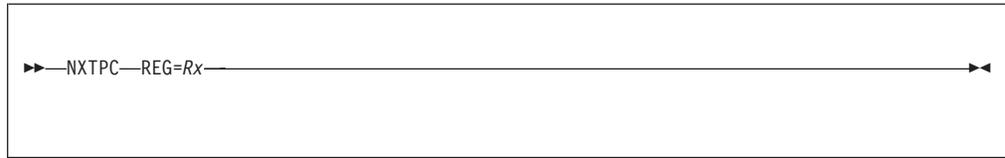
## **Examples**

None.

## NXTPC—Chain Chase through Prefix Pages

Use this system macro to return the real address of the next prefix page.

### Format



#### REG=Rx

Register that contains the real or absolute address of a prefix page. R0 cannot be used.

### Entry Requirements

- The caller must be in 31-bit mode.
- The caller must be executing in a virtual address space; that is, the dynamic address translation hardware (DAT) must be activated.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of specified register will contain the real address of the next prefix page. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- All the prefix pages may be visited using the following procedure:
  1. Load the REG specified register with X'00's indicating the real address of the I-stream prefix page.
  2. Call NXTPC to return a new prefix page address.
  3. Perform whatever prefix page operations are required.
  4. Repeat the calls to NXTPC using the prefix page addresses returned, until NXTPC returns X'00's. This condition indicates that the I-stream prefix page has been looped back to with all prefix pages visited.
- This macro may be processed on any I-stream.
- This macro is for use in the control program (CP) only.

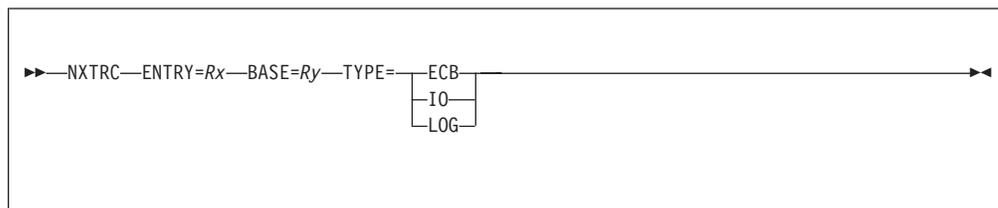
### Examples

None.

## NXTRC—Get next TPF Trace Table Entry

Use this system macro to compute the address of the next available trace entry for the entry control block (ECB) trace table, the TPF log table, and the LDEVBK input/output (I/O) trace table.

### Format



#### ENTRY=*Rx*

On return, the general register specified will contain the address of the requested trace entry. R0 is not valid.

#### BASE=*Ry*

The specified general register will be used to get the base address of the requested trace table. R0 is not valid.

#### TYPE

This parameter identifies the trace table whose entry is to be computed.

##### ECB

Identifies the ECB macro trace table.

##### IO

Identifies the LDEVBK I/O trace table.

##### LOG

Identifies the system log table.

## Entry Requirements

When TYPE=ECB:

- R9 must contain a valid ECB address for the current address space.
- R11 must contain X'1000'.
- R12 must contain X'2000'.
- When TYPE=IO the LDEVBK for the device being traced must be directly addressable through a previously declared USING statement.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- *Rx* contains the address of the assigned trace entry.
- *Ry* contains the base address of the requested trace table.
- The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro should only be used by the appropriate trace monitor routines.
- This macro may be processed on any I-stream.
- When a register is selected as a parameter it must be in the range R1 through R15.

**NXTRC**

**Examples**

None.

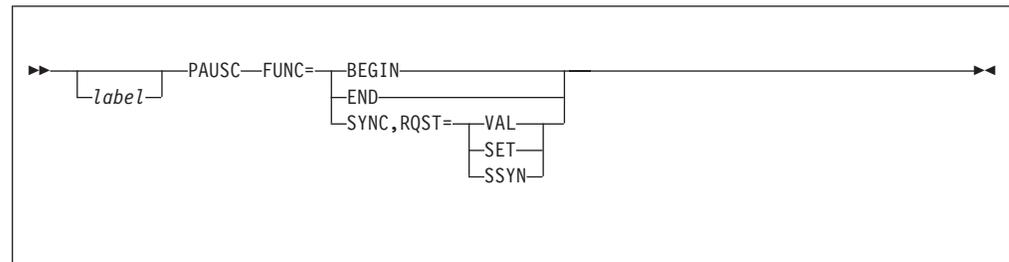
## PAUSC–Control System MP Environment

Use this system macro to enable:

- An E-type program to request that the control program (CP) establish a uniprocessor (UP) environment or reestablish the multiple processor (MP) environment in a TPF system running on a multiple I-stream central processing complex (CPC). This macro can only be run from a program that operates on the main I-stream. This macro enables a program that is not capable of processing in an MP environment to complete its processing in a UP environment.
- The time-of-day (TOD) clock synchronization services to be invoked when a TPF system has been paused.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X	X	

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### **FUNC**

Specify one of the following:

#### **BEGIN**

Establish a UP environment.

#### **END**

Reestablish the MP environment.

#### **SYNC**

Invoke the TOD synchronization services. The TPF system must be in a UP environment at this time.

### **RQST**

The RQST parameter is required when the FUNC=SYNC is specified.

#### **VAL**

Determine and save status of all TOD clocks in the CPC. Restart any stopped clocks.

#### **SET**

Synchronize the main I-stream TOD clock to a requested time. Synchronize all other I-stream TOD clocks to the main I-stream clock value.

#### **Notes:**

1. If the TPF system is in a loosely coupled environment, the main I-stream is synchronized to the external source of clock pulses. This synchronization is via the TOD RPQ or the sysplex timer (STR). All

## PAUSC

other I-streams are synchronized to the main I-stream. If the TPF system is not in a loosely coupled environment, then the main I-stream is either synchronized to a sysplex timer or it remains in local mode. All other I-streams are synchronized to the main I-stream.

2. The IBM 9037 Sysplex Timer is part of IBM Enterprise Systems Connection (ESCON).

### SSYN

Disable sync checks on all I-streams and for the TOD RPQ sets the sync selection register on the main I-stream.

**Note:** If the TPF system is in a loosely coupled environment with the High Performance Option (HPO) feature installed, all I-streams are synchronized to the external source of clock pulses (TOD RPQ or STR). If the TPF system is not in a loosely coupled environment with the HPO feature installed, then the main I-stream remains in local mode, providing clock synchronization pulses for the other I-streams.

## Entry Requirements

- This macro can be run on the main I-stream only.
- R9 must contain the address of the entry control block (ECB) being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of all user registers are preserved across this macro call (R0 through R7, R14 through R15).
- The condition code upon return from this macro is unknown.

## Programming Considerations

- This macro can be run on the main I-stream only.
- Use this macro with discretion. Processing in a UP environment can seriously impact performance on an MP CPC. The time spent in the UP environment must be kept to a minimum.
- The same ECB must stop and restart the MP environment.
- No memory locks or record holds are permitted while the system is in a UP environment (paused).
- The TPF system must be paused by requesting FUNC=BEGIN before any other function request. If the TPF system is not paused, the entry is exited and a system error dump is issued. If another request is active, the entry is exited and a system error dump is taken.
- When requesting a system UP pause, this ECB must not have previously paused the TPF system. If it has, a system error dump is taken and control is returned to the program.
- When requesting a system MP restart, this ECB must have previously paused the TPF system. If not, the entry is exited and a system error dump is issued.

## Examples

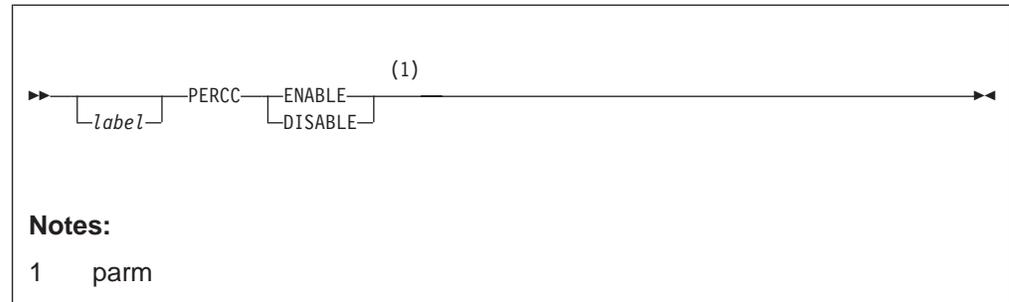
None.

## PERCC–Enable/Disable Program Event Recording (PER)

Use this system macro to enable or disable the TPF system for program event recording (PER).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *parm*

This positional parameter must specify one of the following:

#### **ENABLE**

PERCC sets the PER control bit on in selected program masks and new PSWs.

#### **DISABLE**

The PER control bit is set off in the program masks and PSWs.

See “Return Conditions” for more information about program masks selection.

### Entry Requirements

The ECB-controlled program must be authorized to issue restricted use macros.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The following masks and PSWs are enabled or disabled for PER:
  - SVC New PSW
  - I/O New PSW
  - Selected mask bytes in CAPT at label CPMSMKS
  - The mask byte at CPM1PSW, used during ECB initialization
- The contents of R0 through R7 are preserved.

### Programming Considerations

None.

**PERCC**  
**Examples**

None.

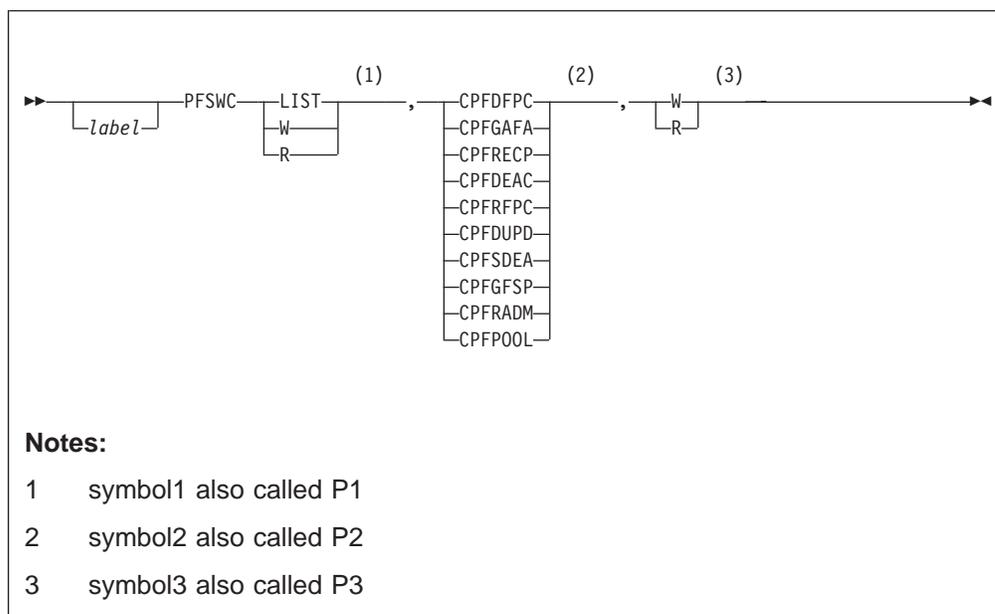
## PFSWC–Reset Pool Function Switch

Use this system macro to generate inline code to reset (that is, set to zero) a pool function switch.

A bit switch is assigned to various mutually exclusive pool functions. A pool function switch is set on (that is, set to 1) by the file pool maintenance and initialization scheduler (CYAA) when activating the associated function. The program performing the pool function resets the switch when appropriate.

These switches provide the means of preventing an operator from inadvertently cycling the TPF system when a pool function is active.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *symbol1*

This positional parameter can be coded as LIST, W, or R.

#### **LIST**

Only a list of equates as required by the file pool maintenance and initialization scheduler (CYAA) is generated

**W** Indicates that storage is unprotected and may be written to.

**R** Indicates that storage is protected and must be unprotected.

#### *symbol2*

This parameter is coded as one of the pool function switches.

#### **CPFDFPC**

Display File Pool Counts.

#### **CPFGAFA**

CRAS-Get File Pool Address.

## PFSWC

### CPFRECP

RECOUP.

### CPFDEAC

Pool Directory Capture.

### CPFRFPC

Reconcile File Pool Counts.

### CPFDUPD

Pool Directory Update.

### CPFSDEA

Son Pool Area Deactivation.

### CPFGFSP

GFS Parameter Program.

### CPFRADM

STA/STP 2314 GFS Randomizer.

### CPFPOOL

Miscellaneous SON Pool Functions.

*symbol3*

Specify one of the following:

**W** Storage is left unprotected.

**R** Storage is left protected.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- For *symbol3*=R or W, the specified pool function switch is OFF (that is, the bit is 0).
- For *symbol3*=R, the storage protect key is set to that of the application core.
- For *symbol3*=W, the storage protect key remains at zero (0).

## Programming Considerations

This macro can be run on the main I-stream only.

## Examples

```
PFSWC LIST
```

This invocation only generates a list of equates.

```
PFSWC W,CPFPOOL,W
```

This invocation indicates that the switch for SON POOLS is unprotected. It is reset and left unprotected upon completion.

```
PFSWC W,CPFPOOL,R
```

This invocation indicates that the switch for SON POOLS is unprotected. It is reset but protected again upon completion.

PFSWC R,CPFPOOL,W

This invocation indicates that the switch for SON POOLs is protected. The switch is unprotected, reset, and left unprotected upon completion.

PFSWC R,CPFPOOL,R

This invocation indicates that the switch for SON POOLs is protected. The switch is unprotected, reset, and made protected again upon completion.

## PHYBC—Return Physical Size of Storage Block

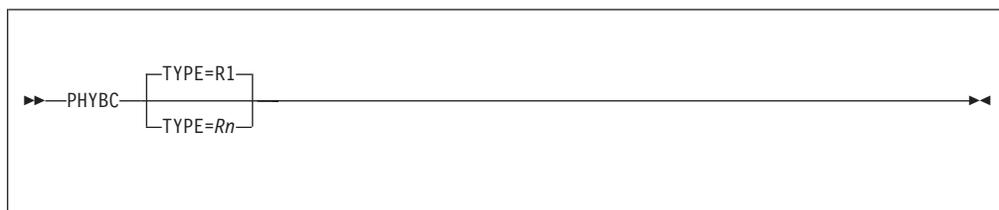
Use this system macro to return the physical size of a storage block.

**Note:** You cannot use all of the available physical storage in a block. You can only use the user size of the block, which is available from the SIZBC macro. The remaining portion of the block is available for system use only. This portion will not be saved on DASD if a block is filed.

See *TPF General Macros* for more information about the SIZBC macro.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### TYPE=R1|Rn

The specified register contains the logical storage block type value. Register R1 is the default value. R0 is not valid for all calls and R8 through R13 are not valid for E-type calls.

Sizes are returned for the following block type equates:

#### LECB (or L3)

Entry Control Block

#### LIOCB

I/O Control Block

#### LSWB

System Work Block

#### LCOMMON

Common frame

#### LFRAME

Frame

### Entry Requirements

- R9 must contain the address of the entry control block (ECB) if this macro is called from a ECB-controlled program.
- For E-type programs R10 must be available as scratch register.
- The specified register must contain the desired storage block type. The valid block types are listed above. These types are defined in the CLHEQ macro.
- This macro can be run in the EVM or the SVM, in either 24-bit or 31-bit addressing mode.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The specified register contains the desired physical storage block size.
- The contents of all other registers are preserved across the macro call (except for R10 on an ECB-controlled program call).
- The condition code is not preserved across the macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- R0 may not be specified.
- This macro is for use in the control program (CP) or an ECB-controlled program.
- The usage of the PHYBC macro requires authorization to issue a restricted macro (CHECK=RESTR) by the \$CKMAC macro.
- A system error dump can occur when servicing a PHYBC macro request. See *Messages (System Error and Offline)* for more information.

## Examples

None.

## PIOFC—Initiate a Preemptive I/O Request

Use this system macro to start an input/output (I/O) request while in preemptive I/O mode. This mode is activated and normal system I/O processing is suspended through the SPNDC macro.

A parameter area, which is called *device operation request block (DOR)* (*DCTDOR*), is used to define the request. The DOR contains fields for information such as the:

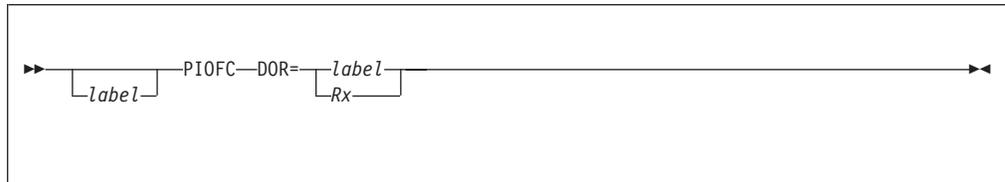
- Address of the channel program
- Device
- Device interrupt handler
- Protect key for the channel program.

A request-unique parameter can be passed to the preemptive input/output (PIO) device interrupt handler upon completion of the I/O using the DOR.

This macro returns a condition code to indicate acceptance of the I/O request. If the request is accepted, the success or failure of the ensuing I/O operation is then determined and handled by the PIO device interrupt handler.

See “SPNDC—Suspend Normal CIO Processing” on page 457 for more information about the SPNDC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**DOR=label/Rx**

This is either a label assigned to the area containing the DOR as defined by the DSECT DCTDOR or a register that contains the address of the DOR.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW key 0.
- The DOR must have the following fields filled in:
 

DORSDA	Symbolic address of the device.
DORCAW	Address of the channel program.
DORKEY	Protect key for operation.
DORLPM	Logical path mask for the operation.
DORFG1	Option flag byte for this request.
DORPRM	Unique request parameter (optional).

DORINT@      Address of device interrupt handler to process interrupts for this request.

## Return Conditions

- Control is returned to the next sequential instruction (NSI) with a condition code as follows.

Condition Code	Meaning
0	Request accepted.
1	Request not accepted because PIO has not been activated by a SPNDC macro. See “SPNDC–Suspend Normal CIO Processing” on page 457 for more information about the SPNDC macro.
2	Request not accepted because a PIOFC request is already active for the specified device.
3	Request not accepted because the specified device is not usable; possible reasons: <ul style="list-style-type: none"> <li>– DORLPM is zero</li> <li>– the SDA is not defined</li> <li>– the SDA is not mounted</li> <li>– the SDA is not operational.</li> </ul>

- R0 through R15 will not be altered.

## Programming Considerations

- This macro can be run on the main I-stream only.
- This macro is for use in the control program (CP) only.

## Examples

None.

## PIORC

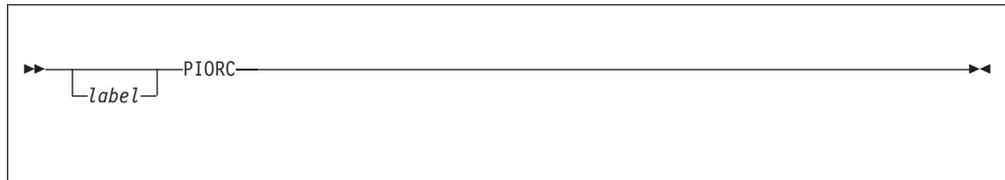
---

### PIORC—Return from PIO I/O Interrupt Processing

Use this system macro, which is issued by a device handler after processing an input/output (I/O) interrupt for a PIOFC operation, to cause control to return to the point of interrupt.

See “PIOFC—Initiate a Preemptive I/O Request” on page 378 for more information about the PIOFC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with the program status word (PSW) key 0.

### Return Conditions

There is no return from the PIORC macro.

### Programming Considerations

- This macro can be run on the main I-stream only.
- This macro is for use in the control program (CP) only.

### Examples

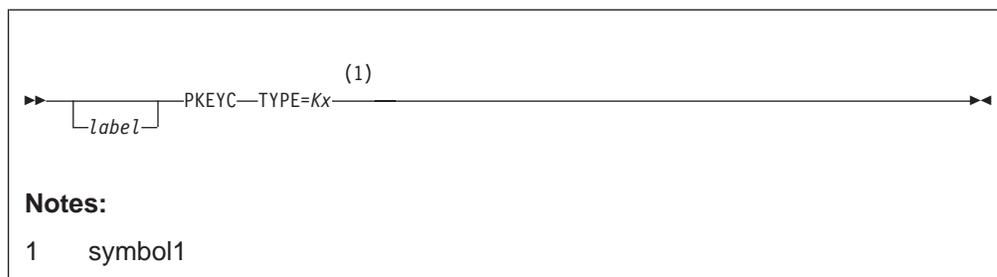
None.

## PKEYC–Keypoint Communication Data

Use this system macro to identify which communication keypoint data is to be prepared for an update to the file-resident copy. The keypointable data is specified in the macro expansion.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

**TYPE=***symbol1*

A symbolic keypoint reference label (K0 through K5) must be specified. The symbolic label *Kx* is used to identify specific keypointable data as indicated by the alphameric value of *x*.

**K0**

Line Status Keypoint

**K1**

Terminal Interchange Status Keypoint

**K4**

Communication Control Unit Keypoint

**K5**

Symbolic Line Status Table Keypoint.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- For a line, terminal interchange, or keypoint directly controlled by the TPF system, the symbolic line number of the communication line must be right-justified in R14.
- For a communication control unit keypoint, the symbolic communication control unit number must be right-justified in R14.
- For a keypoint of the first word of the symbolic line status table (SLST), the symbolic line number must be right-justified in R14.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).

## **PKEYC**

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## **Programming Considerations**

- This macro can be run on the main I-stream only.
- This macro records a request to keypoint communication data residing in host core storage. The request will be honored by the system communication keypoint program.

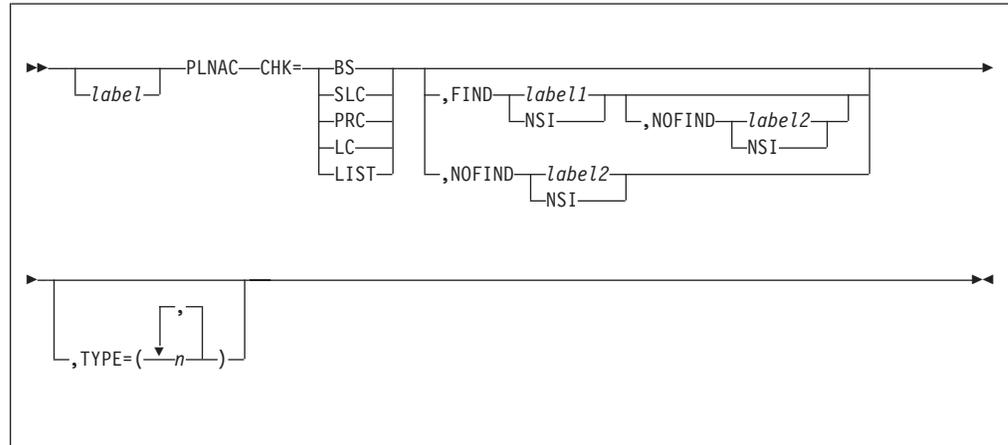
## **Examples**

None.

## PLNAC–Check Symbolic Line Type

Use this system macro to check a symbolic line status table (SLST) entry for one or more specific line types.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### CHK

A symbolic line discipline must be specified. The values and their use are as follows:

#### BS

Check for a BSC line

#### SLC

Check for a Synchronous Link Control line

#### C03

Check for a 2703 controlled line (SLC)

#### PRC

Check for a Prime CRAS terminal

#### LC

Check for a 3270 Local line

#### LIST

Check for a list of line disciplines specified in the TYPE parameter

### FIND

This parameter is optional if NOFIND is coded, but must be coded if NOFIND is not. The possible values are:

*label1*

A user-defined label to be branched to if a requested line discipline is found.

#### NSI

The default of next sequential instruction (NSI) may be explicitly specified.

**Note:** FIND cannot equal NOFIND.

## PLNAC

### NOFIND

This parameter is optional if FIND is coded but must be coded if FIND is not. The possible values are:

*label2*

A user-defined label to be branched to if a requested line discipline is not found.

### NSI

The default of next sequential instruction (NSI) may be explicitly specified.

**Note:** NOFIND cannot equal FIND.

### TYPE

This parameter is required if CHK=LIST. It is a list of line type suffixes to tag CXELTY defined in LINEQ. These suffixes are separated by commas and enclosed in parentheses. The CXELTY tags in LINEQ follow here.

CXELTYn	Description
CXELTY0	RESERVED
CXELTY1	LOCAL 1052-7/3215 CONSOLE-TYPEWRITER
CXELTY9	SLC
CXELTY10	PSEUDO ALC LINE
CXELTY17	BSC
CXELTY18	3272 (LOCAL 3270)

## Entry Requirements

- The communication line equate macro (LINEQ) must be called to resolve the line type labels generated by the PLNAC macro.
- The DSECT macro SLSTL for the symbolic line status table (SLST) must have been called.
- The base register specified on the SLSTL macro must be pointing to the SLST entry to be checked.

## Return Conditions

- Control is returned to the next sequential instruction (NSI) if one of the following occurs:
  - FIND defaults to or is specified as NSI and the line type is found.
  - NOFIND defaults to or is specified as NSI and the line type is not found.
- Control is returned to the FIND label if specified and the line type is found.
- Control is returned to the NOFIND label if specified and the line type is not found.
- The contents of all registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- The storage requirements are summarized in Table 7.

Table 7. Storage requirements for the PLNAC Macro

	FIND or NOFIND but Not Both Specified	Both FIND and NOFIND Specified
CHK=C03	16 bytes	20 bytes
CHK=LIST, TYPE=(t1,...,tn)	n*8 bytes	n*8+4 bytes

Table 7. Storage requirements for the PLNAC Macro (continued)

	FIND or NOFIND but Not Both Specified	Both FIND and NOFIND Specified
CHK=all other options	8 bytes	12 bytes

## Examples

Checking for line disciplines:

Check whether 3270 local line discipline is being used. If it is, branch to a user routine at CPMAC10. If not, branch to a user routine at CPMAC20.

```

ABC      PLNAC  CHK=LC,FIND=CPMAC10,NOFIND=CPMAC20
+ABC     DS      0H              SYMBOLIC LINE TYPE SEARCH
+        CLI    SLSTTYP,CXELTY18
+        BE     CPMAC10
+        B      CPMAC20
+PLNA0077 DS    0H

```

```

CPMAC10 DS    0H
          user routine for FIND

```

```

CPMAC20 DS    0H
          user routine for NOFIND

```

Check for low-speed free-running discipline

Check for line disciplines using TYPE parameter:

Check for line disciplines CXELTY9 and CXELTY17 specified in the TYPE parameter. If either is found, control is transferred to location LABEL10. If neither is found, processing continues with the NSI.

```

CDE      PLNAC  CHK=LIST,FIND=LABEL10,TYPE=(9,17)
+CDE     DS      0H              SYMBOLIC LINE TYPE SEARCH
+        CLI    SLSTTYP,CXELTY9
+        BE     LABEL10
+        CLI    SLSTTYP,CXELTY17
+        BE     LABEL10
+PLNA0079 DS    0H
          next sequential instruction

```

```

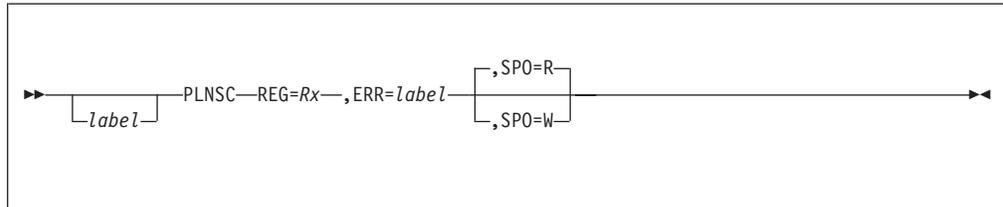
LABEL10 DS    0H
          user routine for FIND

```

## PLNSC—Find SLST Entry

Use this system macro to find the address of a symbolic line number entry in the symbolic line status table (STSTL).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **REG=Rx**

A general register (R0-R7) containing an input symbolic line number.

#### **ERR=label**

This is an error return label that is defined within the user program. A branch to this label will occur if the symbolic line number is not found.

#### **SPO**

Storage protect option.

**R** Allows you only to read the SLST entry. This is the default option.

**W** Allows you to write to the SLST entry.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 and R15 are reserved for use by this macro and cannot be used as the general input register.
- The input register must contain a valid symbolic line number.
- An error routine must be defined within the user program, to handle the error return for a line number that is not valid.

## Return Conditions

- If the line number specified is valid, control is returned to the next sequential instruction (NSI). The general register used as input will point to the proper entry in the SLST table. The USING statement for the SLSTL DSECT will have been issued.
- If the line number is not valid, control is returned to the label specified on the ERR parameter.
- The contents of R14 and R15 are unknown. The contents of R0 through R9 (other than the input register) are preserved across this macro call.
- If the line number is valid, the Write option (SPO=W) will set the storage key to allow modifications of the SLST table.

## Programming Considerations

- This macro can be run on any I-stream.

- The storage requirements depend on the SPO parameter. If the Read (R) option is specified, 34 bytes are required. For the Write (W) option, only 30 bytes are required.
- If the program had issued a USING statement for the SLSTL DSECT prior to using this macro, and the register is not the same as specified in the macro call, DROP the register specified by the PLNSC macro and reissue the USING statement for the previous SLSTL DSECT.

## Examples

None.

## PLONC

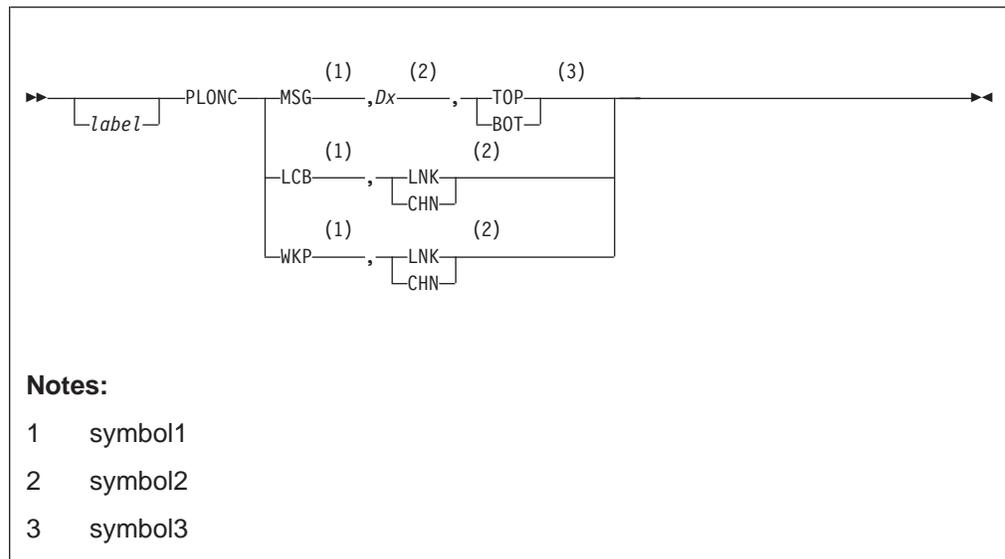
### PLONC–Place on Queue

Use this system macro to:

- Place output messages or link control blocks on CCP queues for transmission on synchronous link lines.
- Restart (wake up) data transmissions by the output CCP after interruption of data transmissions across a link.

**Note:** The link control block (LCB) transmissions had not necessarily been interrupted.

### Format



#### *label*

A symbolic name can be assigned to the macro statement

#### *symbol1*

This identifies the class of PLONC macro, which are:

**MSG**

**LCB**

**WKP.**

#### *symbol2*

If symbol1 is **MSG**, a core block reference word (D0-DF) must be specified as parameter two. If symbol1 is **LCB** or **WKP**, the queue identity (**LNK** or **CHN**) must be specified as parameter two.

#### *symbol3*

This is relevant only if symbol1 **MSG** and specifies that the message block is to be added to the **TOP** or **BTM** (bottom) of the queue.

### Entry Requirements

- If symbol1 is **WKP**, then the symbolic line number of the channel must be right justified in R14.
- If symbol1 is **LCB**, then additionally R15 must contain the link control block.

- If symbol1 is MSG, the message to be sent must be contained in a block of core storage attached to the ECB at the specified level (symbol2).

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of scratch registers (R14, R15) are unknown. The contents of the remaining operational registers and the condition code are saved during processing of this macro.
- If symbol1 is MSG, the specified core block reference word (CBRW) is initialized to indicate that a block of storage is no longer held.

## Programming Considerations

- This macro can only be run on the main I-stream.
- This macro must be used only by synchronous link control (SLC).
- The CAIEQ macro must also be called by the program issuing the PLONC macro so that the assembler may be able to resolve the labels within the PLONC macro expansion.
- The line number is checked to ensure that it is valid and within the legal range for AI lines. If the line number is out of range, control is transferred to the system error routines and the ECB is forced to exit.
- If symbol1 is MSG, a check is made by the control program (CP) to determine if the ECB is holding a block of storage at the specified level. If a block is not held, control is transferred to the system error routines and the ECB is exited.

## Examples

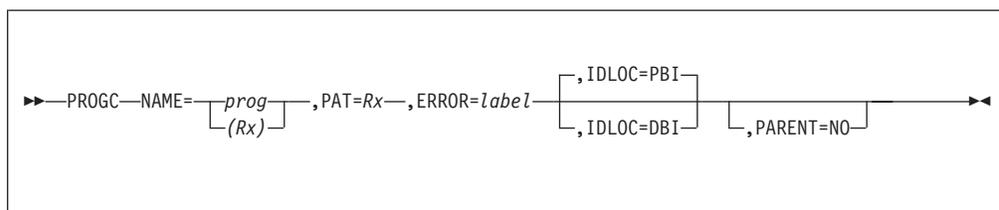
None.

## PROGC—Return Program Information

Use this system macro to return the address of a program’s program allocation table (PAT) entry. This macro searches the PAT for the program name specified and, if a match occurs, returns the address of the program’s entry in the PAT. If the program name specified is not found in the PAT, no address is returned and the macro branches to the label specified in the ERROR parameter of the macro.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### NAME=prog(*Rx*)

This parameter specifies the name of the program to be located. The name is 4 alphanumeric characters that should have been allocated at system generation time.

#### *prog*

The name of the program whose PAT entry is to be located.

#### (*Rx*)

A register (R0 through R7) that contains the name of the program whose PAT entry is to be located.

#### PAT=*Rx*

This parameter specifies the general register (R0 through R7) that will contain the address of the requested program’s PAT entry on output.

#### ERROR=*label*

This parameter, a symbolic name, specifies the location to which control will be given if no match is found.

#### IDLOC

Specifies whether the request will be serviced using the program base ID or the database ID.

#### PBI

The request will be serviced using the program base ID located in CE1PBI. This is the default.

#### DBI

The request will be serviced using the database ID located in CE1DBI.

#### PARENT=NO

Specifies that the request will return the transfer vector PAT address if the program is a transfer vector. Otherwise, the PAT address of the parent is returned.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI) if the program name was found. If the program name was not found, control is transferred to the label specified by the ERROR parameter.
- The general register specified on the PAT parameter contains the PAT address of the requested program if the program was found or zero if the search was unsuccessful.
- The contents of R0 through R7 are preserved with the exception of the register specified on the PAT parameter.
- If the program name specified is a transfer vector, the address of the PAT entry of the parent program will be returned unless PARENT=NO is coded. For this condition, the address of the transfer vector PAT entry will be returned.

## Programming Considerations

- This macro is restricted for use by E-type programs only. C-type programs can issue the PROGC macro to generate a DSECT for the PROGC parameter list.
- If the program specified is I-stream unique, the PAT entry address returned will be the program's PAT entry unique to the I-stream from which the macro was invoked.

## Examples

None.



**LEVEL=*Dx***

The data level of the IDSAOS block. Acceptable levels range from D0 to DF.

**ERROR=*label***

The label of a routine where processing is to continue in the event of an error condition.

## Entry Requirements

- A valid token must exist in the ECB field CE1TKN for the QUERY and CLOSE functions.
- A CLOSE function must be performed before a second CREATE may be issued by an ECB.

## Return Conditions

One of the following return codes will be provided in R0.

<b>Return Code</b>	<b>Meaning</b>
0	Successful completion.
4	Token already active.
8	The asynchronous event table is full.
12	Operations pending for a token.
16	ECB does not own token.
20	A token number that is not valid. This token number is 0.
24	Core block (IDSAOS) full.

## Programming Considerations

- This macro can be run on any I-stream.
- Specifying RETURN=PENDING is not valid when WAIT=YES is specified.
- LEVEL, WAIT and RETURN parameters can only be specified when FUNC=QUERY is specified.
- The LEVEL parameter is required when FUNC=QUERY is specified.
- A data block, mapped by IDSAOS, must be attached on the specified data level to return status information for FUNC=QUERY.

## Examples

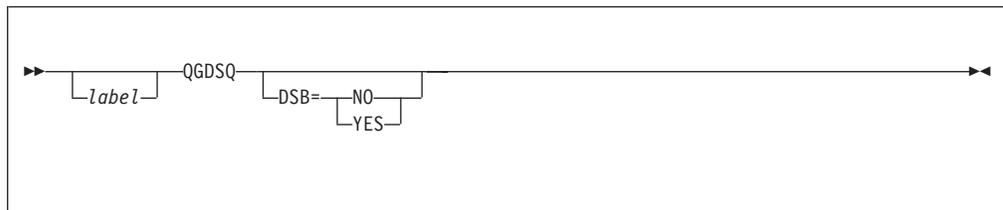
None.

## QGDSQ—Query General Data Set (GDS) Input/Output (I/O) Queue

Use this system macro to determine the number of input/output (I/O) requests that are queued to the device that is associated with a general data set (GDS) mounted to the TPF system.

Required Authorizations			
Key0	Restricted	System	Common Storage
		X	

### Format



#### *label*

An optional label can be used with this macro.

#### **DSB**

This parameter specifies if the first data set control block (DSCB) address that is associated with the input data set name will be returned in register 1 (R1) when macro processing is completed successfully.

#### **NO**

This parameter indicates that the first DSCB address is not returned to the caller.

#### **YES**

This parameter indicates that the first DSCB address is returned to the caller.

### Entry Requirements

- This macro can be run only by ECB-controlled programs.
- R1 must contain the address of the 16-character data definition (DD) name to query.

### Return Conditions

- If the queue is queried successfully, R0 contains the number of requests that are queued to the device that is associated with the input DD name. R1 contains the address of the first DSCB associated with the data set if you specify YES for the DSB parameter.
- If the queue is not queried successfully, R1 contains -1.

### Programming Considerations

If multiple data sets are on the device, the queue count that is returned includes all queued requests for all data sets on the device.

## Examples

In the following example, a QGDSQ macro request is made to get the number of I/O requests that are queued for a device that is associated with an input data definition (DD) name of 'INPUTDDNAME'. The first DSCB that is associated with the data set is returned to the caller.

```

:
LA    R1,INPUTDD           Point to DDNAME for data set
QGDSQ DSB=YES             Check DDNAME and count of requests
LTR   R0,R0               Successful return?
BM    NOT MOUNTED        No, data set not mounted
BZ    NO_REQUESTS        Yes, but no queued requests

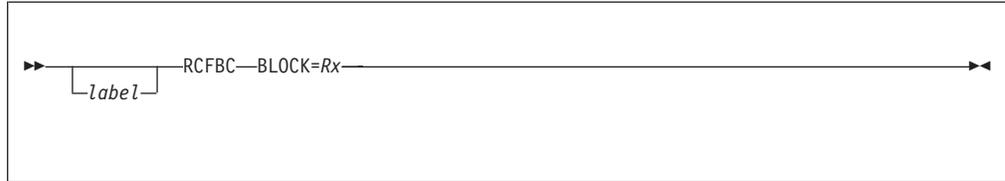
                               Otherwise I/O requests queued...
                               R0 contains the count of requests
                               R1 points to first DSCB for data set
:
INPUTDD DC CL16'INPUTDDNAME' DDName to query
:

```

## RCFBC—Release Coupling Facility Work Block Address

Use this system macro to release the coupling facility work block.

### Format



*label*

specifies a symbolic name that can be assigned to the macro statement.

**BLOCK=Rx**

specifies the address of a coupling facility work block to be released, where *Rx* is a register from R0 to R7.

### Entry Requirements

None.

### Return Conditions

Control is returned to the next sequential instruction (NSI).

### Programming Considerations

- This is a restricted use macro.
- This macro can be processed on any I-stream.

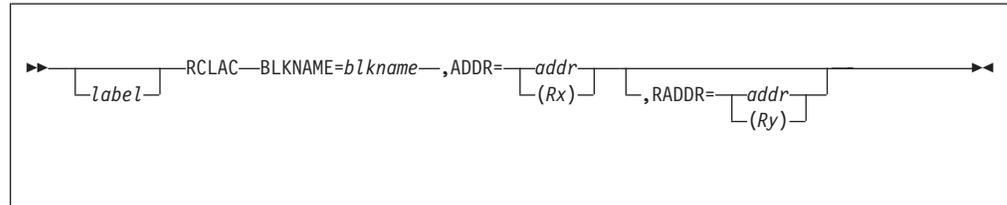
### Examples

None.

## RCLAC—Release a Specified CLAW Block Type

Use this system macro inline to release addressability to the Common Link Access to Workstation (CLAW) control blocks.

### Format



*label*

A symbolic label can be assigned to this macro statement.

**BLKNAME=blkname**

The name of the block being returned. BLKNAME is a required parameter.

**FOURKF**

Locked page for channel control word (CCW)

**ICADAP**

Adapter control block

**ICLAWB**

CLAW device interface block

**ICLAWG**

CLAW page structure

**ICLCON**

Connection control table

**ICLIBK**

Client control block

**ICLIOI**

Extension block for CLAW I/O interrupt

**ICLTRB**

Transaction control block

**ICMSGB**

Message control block

**ICNBLK**

Extension block for CLAW initialization

**ICPATH**

Path control block

**ICPERM**

Permanent work area

**ICPOLL**

POLL request extension block

**ICQBLK**

Message queue element structure

## RCLAC

### ICRBLK

Extension block for returning CLAW page

### ICRCCW

Read channel CCW area

### ICTRCE

Trace data structure

### ICWCCW

Write channel CCW area

### ISCCDT

CLAW device table

### ISCFDT

File descriptor

### ISCIPT

Internet Protocol (IP) address table.

### ADDR

The system virtual memory (SVM) address of the returned block. ADDR is a required parameter.

*addr*

An address.

(Rx)

A register containing the address.

### RADDR

The SVM address of the returned block. RADDR is an optional parameter.

*addr*

The address.

(Ry)

A register containing the address.

## Entry Requirements

This macro is for use in the control program (CP) only.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of all registers, except R14 and R15, are preserved across this macro call.

## Programming Considerations

This macro can be run on any I-stream.

## Examples

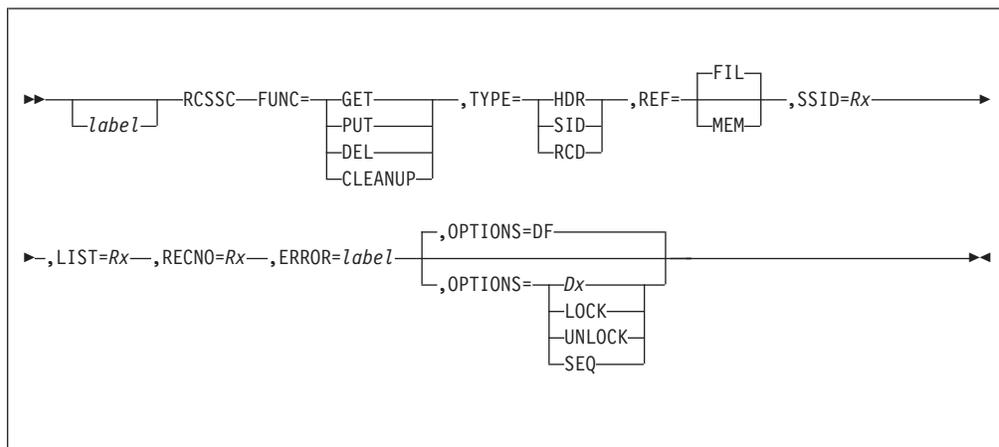
- This call releases block ICTRCE located at SVM address VCPTRCBK.  
RCLAC BLKNAME=ICTRCE,ADDR=VCPTRCBK
- This call releases block ICLTRB located at the address found in register R10.  
RCLAC BLKNAME=ICLTRB,ADDR=(R10)



## RCSSC—Access the Record Cache Subsystem Status Table

Use this system macro to access the record cache (RSC) subsystem status table (SSST) and make updates to the it. Using this macro insulates routines that process the RCS SSST from changes in the data structure of the table.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **FUNC**

The function to be performed. This parameter is required.

#### **GET**

Retrieve SSST information

#### **PUT**

Update SSST information

#### **DEL**

Invalidate SSST information

#### **CLEANUP**

Release SSST information

#### **TYPE**

The type of data requested. This parameter is required.

#### **HDR**

Header

#### **SID**

SSID (subsystem status ID) entry

#### **RCD**

File record

#### **REF**

The SSST structure to be used for the request. This parameter is required.

#### **FIL**

Read in SSST block from DASD and establish pointers to the memory and file structures. The default is FIL.

#### **MEM**

Establish memory pointers only. CP segments must code REF=MEM.

**SSID**

The record cache subsystem status identifier (SSID) value for the request. This parameter is required.

*Rx* Register containing the SSID value. Valid values range from R0 through R15.

**LIST**

The RCSSC macro parameter list (IDSCS0) pointer. This parameter is required. The RCSSC macro uses IDSCS0 to pass requested information to the service routines. The RCSSC macro generates the code necessary to build the parameter list and invokes the CP and real-time service routines.

*Rx* Register containing the address of the request parameter list (IDSCS0). The parameter list will be filled out by the RCSSC macro. Valid values range from R0 through R15. The default is R1.

**RECNO**

The file SSST ordinal number for the request (required only when TYPE=RCD is specified):

*Rx* Register containing the file ordinal. Valid values range from R0 through R15.

**ERROR**

The label of a routine where processing continues on an error condition.

*label*

Label of a program statement that gains control on an error return from RCSSC.

**OPTIONS**

The processing options associated with the request:

*Dx* Use data levels D0 through DF for file operations. DF is the default.

**LOCK**

Lock IDSSST file data structure.

**UNLOCK**

Unlock IDSSST file data structure.

**SEQ**

Access data type in sequential mode.

## Entry Requirements

The register to point to the parameter area using the LIST parameter. R1 is the default. The macro itself will fill in the parameter area with the values coded on the macro call.

## Return Conditions

- Control is returned to the next sequential instruction (NSI) unless the ERROR parameter is coded and an error condition occurs.
- A parameter list is set up to access the requested portions of SSST. A parameter list return code is set if errors are detected.
- The return code modifier field of the parameter list (IDSCS0) should be examined on each return from the RCSSC macro when using SSID requests. When the SSID does not exist, the macro will set the EOF condition and return with the slot pointer pointing to the first available overflow slot. If this slot is to be filled in, the SSST header needs to be updated to reflect the new count of active slots.

## RCSSC

### Programming Considerations

- This macro can only be run on the main I-stream.
- FUNC, TYPE and REF are required parameters.
- RECNO parameter is required only when TYPE=RCD is specified.
- Registers R0 through R7 will be saved and restored by the macro. Other registers can be used for the LIST, RECNO, and SSID parameters but they may be corrupted by the macro. All registers are saved during control program (CP) calls.
- The options LOCK and UNLOCK are mutually exclusive.
- CP programs are restricted to memory requests only. ECB programs may specify memory or file requests.
- REF=FIL must be specified when updating the SSST.
- All updates must be made in the file entry. The macro will reflect the changes in the memory entry on the subsequent PUT request.
- When using the CLEANUP function, the UNLOCK option must be specified and REF=FIL must be coded as it was on the RCSSC macro that retrieved the lock.
- SSID parameter is required with TYPE=SID unless the SEQ option is specified. The SSID parameter cannot be used with the SEQ option.
- The macro processor will try to use the SSID passed with this parameter as the starting point of the sequential search but the parameter list will not be set up with the pointers to this SSID in the SSST.
- The SEQ option is used to process the entire SSST structure.
- You must clear the CS0SSID field of the RCSSC parameter list before the first RCSSC invocation. This field will be used by the macro processor on subsequent calls as the starting point to scan for the next active SSID.
- In general, for CP segments, R1 is used to pass information to the RCSSC service and R0 is used to pass return code information to the user. The exception to this is for a header request where the address of the RCSSC header is returned in the register specified by the LIST parameter.
- The SSST is a basic subsystem (BSS) resident table. All RCSSC calls must be made from the BSS.

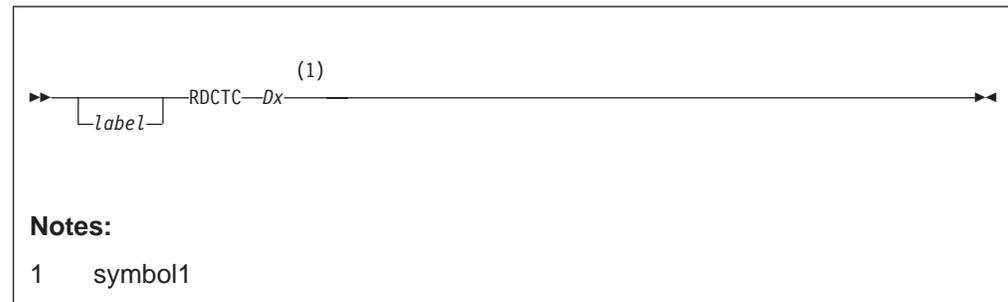
### Examples

None.

## RDCTC–3705 Communications

Use this system macro to process a chain of user-defined channel command words (CCWs) so you can communicate with a 3705 Communications Controller over its native subchannel (NSC).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*symbol1*

A file address reference word (D0 to DF) must be specified for this parameter.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A 3705 native subchannel address (SIO address) and the core location of the first user-defined CCW must be contained in the FARW for the specified level as follows.

CE1FAX	Bytes 0-3	Storage address of first CCW
CE1FMx	Byte 2	Channel address
	Byte 3	NSC address

- You must define all fully constructed CCWs needed.
- CCW operation codes must be valid for the 3705.
- Storage locations in which data is accessed are your responsibility.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The status of the operation is unknown.
- The contents of the core block reference word at the specified level are unchanged.
- Byte 0 of the file address reference word is cleared; the remainder is unchanged.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on the main I-stream only.
- A check is made by the TPF system to determine that the NSC is valid.
- No check is made to verify the validity of the CCW operation codes.
- The operational program may not use the specified level until the operation is complete.

## RDCTC

- A WAITC macro must be issued to ensure completion of the operation. The control program (CP) will transfer control to the user-defined CCWS and handle the following error conditions:
  - Equipment or Bus Out Check
  - Intervention Required or CA Not Initialized Indication
  - Abort.

On uncorrectable error conditions the gross error indicator and the detailed error indicator for the specified level will be set, and the channel status word will be inserted into the file address reference word for the specified level.

- This macro is restricted to use by the 3705 load and dump programs.

## Examples

```
LABEL   RDCTC  D2
```

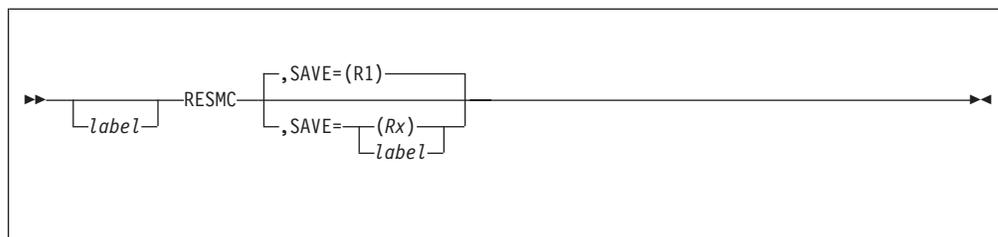
## RESMC–Resume Normal CIO I/O Processing

Use this system macro to enable the user of the preemptive input/output (I/O) (PIO) facility to resume normal system I/O processing (CIO). When the user is finished with PIO, the TPF system issues this macro specifying the registers to be loaded and the program status word (PSW) to be used when passing control back to normal system operation.

Before normal system operation is resumed, PIO dispatches the queued interrupts to the CIO device interrupt handlers. These are the interrupts received by PIO for I/O operations initiated by CIO before normal system operations were suspended.

If the resume PSW shows that the TPF system is disabled for interrupts, the queued interrupts are not dispatched. PIO sets a trap to determine when the TPF system becomes enabled at which time the queued interrupts are dispatched.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### SAVE

This is either a label assigned to the save area or a register other than R0 that contains the address of the area. If SAVE is not coded, the area address defaults to R1.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW key 0.
- The save area, containing the information required to resume normal TPF processing, must have the following format:

Words 1-2      The resume PSW.

Words 3-18    Values to be reloaded into R0 through R15.

### Return Conditions

There is no return from processing this macro.

### Programming Considerations

- This macro can be run on the main I-stream only.
- This macro is for use in the control program (CP) only.

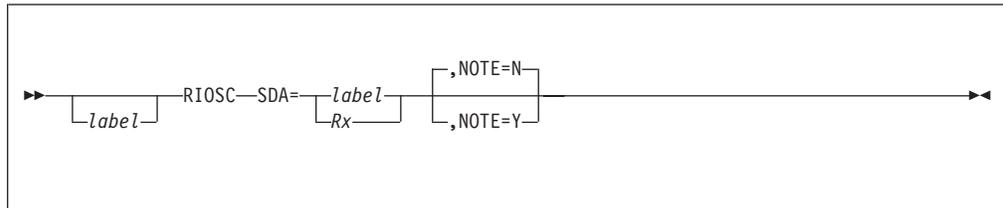
### Examples

None.

## RIOSC—Reset an I/O Operation

Use this system macro to issue the necessary instructions to halt and clear any input/output (I/O) for a specific symbolic device address (SDA). The SDA is available for further I/O use upon return from this macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**SDA=label|Rx**

This is either a label of a halfword field containing the SDA or a register which contains the SDA in bytes 2 and 3 and zeros in bytes 0 and 1.

#### NOTE

This denotes whether a notification interrupt is to be created and presented to the device handler if an active I/O request is cleared. The notification interrupt consists of an interrupt with device status of zero and interface control check set in channel status.

**N** No notification is to be generated (default)

**Y** Notification interrupt is to be generated.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW Key 0.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of all registers are preserved across this macro call.

### Programming Considerations

- This macro must be processed only on an I-stream having affinity with the specified SDA.
- RIOSC cannot be processed from a real-time program.

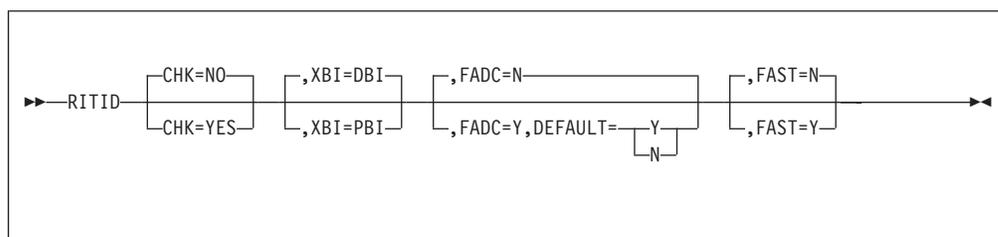
### Examples

None.

## RITID–Access RIAT Entry

Use this system macro to access a specific record identifier (ID) in the record ID attribute table (RIAT). If the specified record ID is in the RIAT, this macro moves the RIAT record to the caller's work area. If the specified record ID is **not** found in the RIAT, this macro sets the address of the RIAT section in the caller's work area to zero (the DCTRIT RITADDR field).

### Format



#### CHK

Parameter used for LEBIC call. This parameter is valid only when RITID is invoked by a CP segment.

#### YES

Integrity checking is required on the LEBIC macro call.

#### NO

Integrity checking is not required on the LEBIC macro call.

#### XBI

Program/data base index, used by LEBIC macro. The default is DBI. This parameter is valid only when RITID is invoked by a CP segment.

#### DBI

CE1DBI is to be used as first parameter in LEBIC macro.

#### PBI

CE1PBI is to be used as first parameter in LEBIC macro.

#### FADC=N|Y

FADC=Y is used by the CP file access code to generate module specific RIAT support code. The default is FADC=N. FADC=Y is restricted to the control program (CP).

**Note:** The interface requirements when this parameter is specified as Y are module specific.

#### DEFAULT=Y|N

DEFAULT=Y or DEFAULT=N along with FADC=Y is used by the CP file access code to generate module specific RIAT support code. DEFAULT has no meaning when FADC=Y is not coded.

#### FAST=N|Y

Specifies whether control program (CP) routines are used for processing. This parameter is ignored unless used within the control program (CP) and FADC=1. The default is FAST=N.

## Entry Requirements

There are two sets of entry requirements:

## RITID

1. The first set, the work area option, can be used by real-time programs or by the control program. R1 points to the caller's work area. The minimum length of the work area must be RITIDLN as defined in DCTRIT. The RITRID field in the work area must be set to the record ID to be searched for in the RIAT. This ID must be in hexadecimal format (for example, X'1355') or character format (for example, C'AS').
2. The second set, the FADC option, can be used only by the control program. The CHK, XBI, DEFAULT, and FAST parameters apply to the FADC option.

## Return Conditions

1. Work area option

The work area passed by the caller will contain the RIAT item. The RIAT item in the work area maps to the RTID DSECT (contained in DCTRIT). RITADDR will contain the address of the item in the RIAT table.

If the specified ID is not found in the RIAT, the address of the ID entry in the RIAT (accessed by field RITADDR in the RTID DSECT) in the work area will be set to zero. The RIAT item in the work area will contain all the default values located in the RIAT header.

Registers 0 to 15 are preserved across this macro call.

2. FADC option

The appropriate fields in the MIOB block will be set.

## Programming Considerations

The RITID macro can be processed on any I-stream.

## Examples

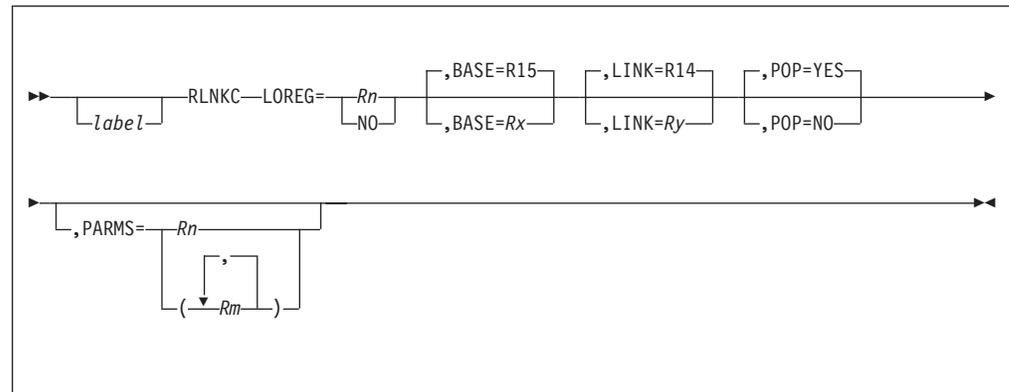
None.

## RLNKC—Return to CP Calling Routine and Reset Stack Pointer

Use this system macro to reload the linkage data from the stack, reset the stack pointer, and return to the calling control program (CP) routine. This macro is used with other standardized linkage macros such as the CLNKC, DLNKC, and SLNKC macros. See the following for more information about these macros:

- “CLNKC—Control Program (CP) Call and Link” on page 166
- “DLNKC—Define Stack DSECT for Control Program (CP) Routine” on page 216
- “SLNKC—Control Program (CP) Save Link Data & Set Stack Pointer” on page 448.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **BASE=R15|R<sub>x</sub>**

The register specified, default R15, was used as the link-to register by the calling routine (CLNKC).

#### **LINK=R14|R<sub>y</sub>**

The register specified, default R14, will be used as the link register from this routine. The calling routine used this register as its link-from register.

#### **LOREG=R<sub>n</sub>|NO**

Required. If a register *R<sub>n</sub>* is specified, all registers, beginning with the specified register, will be reset from the stack. If NO is specified, no registers are reset from the stack.

#### **POP**

Specify one of the following:

##### **YES**

The stack register will be popped, that is, adjusted by the length of this routine’s stack area.

##### **NO**

The stack register will not be ‘popped,’ since there is no stack area used by this routine and it does not call any routine.

Used in conjunction with the PUSH parameter on the SLNKC macro.

#### **PARMS=R<sub>n</sub>|(R<sub>m</sub>,...,R<sub>n</sub>)**

This is optional. The register (or registers) specified by this parameter will *not* be reset from the stack since it is assumed to contain data.

## RLNKC

### Entry Requirements

- R13 must contain the stack address.
- The LINK= register must contain the address of the link-from routine.

### Return Conditions

The macro generates code to load the appropriate registers from the stack, excluding the parameter registers. It adjusts the stack register by the size of the stack area for this routine and returns to the calling routine via the LINK register.

### Programming Considerations

- This macro can be run on any I-stream.
- R13 must point to a valid stack area.
- This macro, in conjunction with the SLNKC macro, is used to manipulate the stack register.
- The condition code setting is determined by the interface defined for the routines using the linkage macros.
- The LOREG parameter must match with the corresponding parameter on the SLNKC macro. If they do not match, the LOREG parameter is used on the RLNKC macro.
- The POP parameter must match with the corresponding PUSH parameter on the SLNKC macro. If they do not match, the POP parameter on the RLNKC macro is used, but errors may occur if there is a mismatch on R13 usage.
- This macro is for use in the control program (CP) only.

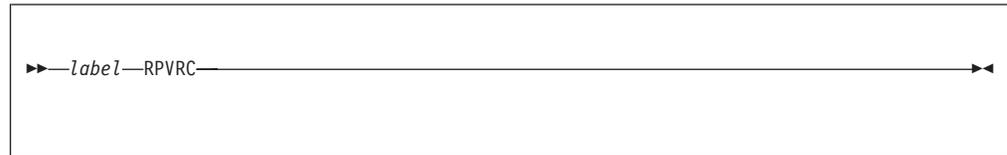
### Examples

None.

## RPVRC—Read and Process Program Version Record

Use this system macro to read all the program version records (PVRs) from file into main storage and to add the appropriate information from each PVR entry to the corresponding program allocation table (PAT) entry.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

The RPVRC macro is restricted to ECB-controlled programs.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- All registers are preserved across the macro call.

### Programming Considerations

If unable to process all the PVRs for a subsystem, then state change will be disabled for the subsystem.

### Examples

The following example shows the RPVRC macro being issued for each subsystem in the complex.

```

ICM R14,3,CE1DBI
CEBIC DBI,S
CEBIC BSS                                RESTORE DBI AND SSU ID TO BSS
CTKSPVR DS 0H

RPVRC                                    PROCESS PVRs FOR THIS SS
LEBIC DBI,R4,CHECK=NO                    GET SS ID WITHOUT COMPLEMENT
PVRSSID DS 0H
LA R4,1(,R4)                             NEXT DBI
LR R6,R4

UATBC IDLOC=(R,SSI,R6),                  SS ID
      EXCD=PVRDONE,                       FINISHED
      INVLID=PVRSSID,                     NEXT
      NOTAVL=PVRSSID                     NEXT

MSOUT REG=R6
ICM R14,3,MU0DPI                          LOAD SS ID WITH COMPLEMENT

CEBIC DBI 07240000
B CTKSPVR
PVRDONE DS 0H                             FOUND - DO NEXT SS
                                           FINISHED - CONTINUE

```

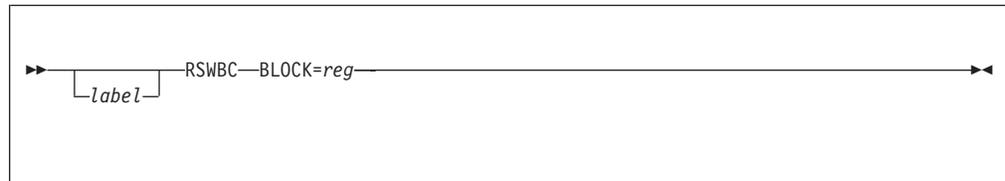
## RSWBC

### RSWBC—Release a System Work Block (SWB)

Use this system macro to release a system work block (SWB) address.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=reg**

Specifies the address of the SWB to be released, where *reg* is a register from R0 – R7.

### Entry Requirements

This macro is restricted to ECB-controlled programs.

### Return Conditions

None.

### Programming Considerations

None.

### Examples

The following example releases the SWB specified in register 5 (R5).

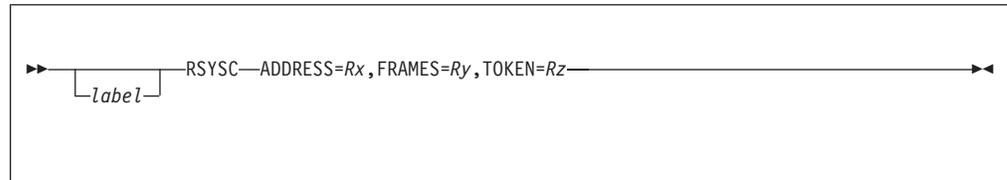
```
RSWBC BLOCK=R5
```

## RSYSC—Release System Heap Storage

Use this system macro to return frames to the TPF system that were allocated to the system heap by the \$GSYSC or GSYSC macros.

See “\$GSYSC—Get System Heap Storage” on page 50 and “GSYSC—Get System Heap Storage” on page 279 for more information about the \$GSYSC and GSYSC macros, respectively.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### **ADDRESS=R<sub>x</sub>**

The ADDRESS parameter specifies the register containing the starting address of the storage being returned. This address must be:

- A valid system heap address
- On a 4 KB boundary.

The general register used must be R0 through R12, R14, or R15.

#### **FRAMES=R<sub>y</sub>**

The FRAMES parameter specifies the number of 4 KB frames to be returned to the TPF system. The number of frames requested for release must be the same number of frames requested on the \$GSYSC or GSYSC macro. The general register used must be R0 through R7, R14, or R15.

#### **TOKEN=R<sub>z</sub>**

The TOKEN parameter specifies the address of an 8-character string that the TPF system uses to identify the allocated storage. This string must match the string specified on the \$GSYSC or GSYSC macro. The general register used must be R0 through R7, R14, or R15.

### Entry Requirements

- The general register indicated by ADDRESS must contain the starting address of storage allocated using the GSYSC or \$GSYSC macro (or the C language functional equivalent).
- R9 must contain the address of a valid ECB.

### Return Conditions

- The TPF system always returns control to the application.
- R15 contains the following return codes.

#### **Return Code**

#### **Meaning**

0

The TPF system has successfully released all specified storage.

RSYSC\_ERROR

The address specified is not valid, the storage is

## RSYSC

not in use, and the starting address, size, and token do not match current storage allocations. No storage is released.

- All registers except R15 are preserved.

## Programming Considerations

- This macro must use the same FRAMES and TOKEN parameters as specified on the GSYSC or \$GSYSC macros. See “GSYSC—Get System Heap Storage” on page 279 and “\$GSYSC—Get System Heap Storage” on page 50 for more information about the GSYSC and \$GSYSC macros, respectively. For example, if 12 KB of storage is allocated with a token of TABLE, the release macro must be coded to release 12 KB of storage with a token of TABLE.
- When the macro successfully completes processing, the macro trace entry contains the size (in number of frames) and address of the allocated area in addition to the normal macro trace information.
- The \$RSYSC macro is used only by the control program (CP) while this macro is used by application programs. See “\$RSYSC—Release System Heap Storage” on page 81 for more information about the \$RSYSC macro.

## Examples

Return system storage allocated for use as MY\_TABLE.

```
          ITUUTL REG1=R14           CONNECT WITH TABLE UPDATE DSECT
          LA     R14,ITULEN         GET THE LENGTH OF A BLOCK
          LA     R14,4095(R14)      ROUND TO THE NEXT 4 KB
          LR     R7,R14             SAVE NUMBER OF FRAMES
          SRL   R14,12             DETERMINE NUMBER OF 4 KB FRAMES
          LA     R6,MY_TABLE
          GSYSC FRAMES=R14,TOKEN=R6 ALLOCATE THE STORAGE
          LTR   R14,R14            CHECK THE RETURN CODE
          BNZ   HAVE_STORAGE       CONTINUE PROCESSING
          .
          .
          process error for no storage
          .
          HAVE_STORAGE    DS 0H
          .
          .
          routine that uses the storage
          .
          .
          RELEASE_STORAGE DS 0H
          LA     R6,MY_TABLE
          RSYSC ADDRESS=R14,FRAMES=R7,TOKEN=R6  RELEASE THE STORAGE
          LTR   R15,R15            CHECK THE RETURN CODE
          BNZ   RELEASE_ERROR     BRANCH TO PROCESS ERROR
          .
          .
          .
          MY_TABLE  DC  CL8'MY_TABLE'
```

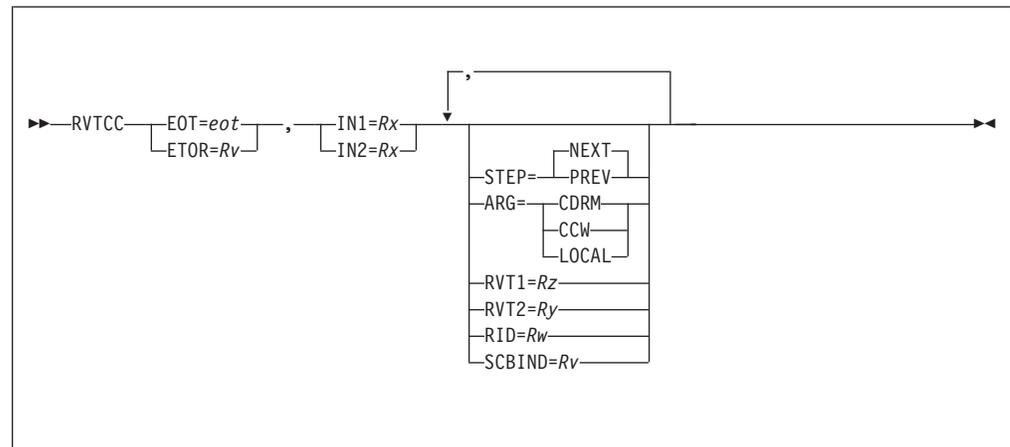
## RVTCC–Search RVT Entries

Use this system macro to compute the RVT1 or SCB1 address, the RVT2 or SCB2 address, and the resource identifier (RID) or SCBID for the next or previous resource in the hierarchy of the network configuration.

**Note:** If you specify an SCB1 or SCB2 address, this address must reference a session control block (SCB) that is anchored off an entry in the resource vector table (RVT).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### EOT=eot

This parameter specifies that a branch is to be taken to the label *eot* if the input or stepped entry is the delimiting entry of the table.

### EOTR=Rv

This parameter specifies that a branch is to be taken to the address specified in register *Rv* if the input or stepped entry is the delimiting entry of the table.

### IN1=Rx

This indicates that the address passed in register *Rx* as input to the macro is an RVT1 or SCB1 address. If it is an SCB1 address, it must reference an SCB that is anchored to the RVT.

### IN2=Rx

This indicates that the address passed in register *Rx* as input to the macro is an RVT2 or SCB2 address. If it is an SCB2 address, it must reference an SCB that is anchored to the RVT.

### STEP

This parameter specifies which resource in the table is to be retrieved.

### NEXT

The next resource in the table is to be retrieved.

### PREV

The previous resource in the table is to be retrieved.

## RVTCC

### ARG

This parameter can be used to key the searching on a particular RVT entry value.

### CDRM

When STEP=NEXT, the entry of the next resource owned by the CDRM is retrieved. When STEP=PREV, the entry for the CDRM is retrieved.

### CCW

When STEP=NEXT, the entry for the next resource whose session runs through the ALS/NCP/CTC is retrieved. When STEP=PREV, the entry for the ALS/NCP/CTC is retrieved.

### LOCAL

When STEP=NEXT, the entry of the next local resource is retrieved. When STEP=PREV, the entry for the local SSCP is retrieved, with one exception: if the input entry is an SLU thread, the entry of the application PLU is retrieved.

If you do not specify the ARG parameter, the next (previous) resource of any type is retrieved.

### RVT1=Rz

Register *Rz* contains the RVT1 or SCB1 address for the stepped resource on return.

### RVT2=Ry

Register *Ry* contains the RVT2 or SCB2 address for the stepped resource on return.

### RID=Rw

Register *Rw* contains the RID or SCBID of the stepped resource on return.

### SCBIND=Rv

Register *Rv* contains an indicator of whether the values returned through RVT1, RVT2, and RID reference an RVT or an SCB. If they refer to an SCB, *Rv* contains one (X'00000001'); if they refer to an RVT, *Rv* contains zero.

## Entry Requirements

- The input register specified by IN1 or IN2 must contain a valid RVT1 (or SCB1) or RVT2 (or SCB2) address, respectively.
- Operands specifying input registers can use R0 through R7, R14, or R15.

## Return Conditions

- The registers specified in the macro call are set appropriately.
- Processing continues following the macro expansion, unless a delimiting entry is found. In that case, a branch is taken to the label specified by the EOT parameter or to the address in the register specified by the EOTR parameter.
- Delimiting entries are detected under the following conditions:
  - For all entry conditions:
    - When the entry retrieved is the first entry in the table
    - When the entry retrieved is the end of table entry.
  - For STEP=NEXT and any ARG settings:
    - When the input entry is either the last RVT (if there are no SCBs anchored off it) or the last SCB in the chain anchored off the last RVT.
  - For STEP=PREV and any ARG settings:

- When the input RVT entry is the first entry in the table.
- For STEP=PREV and ARG=CDRM:
  - When the entry for the CDRM cannot be found.
- For STEP=PREV and ARG=CCW:
  - When the entry for the ALS/NCP/CTC cannot be found.
- For STEP=PREV and ARG=LOCAL:
  - When the entry for the local SSCP cannot be found
  - When the entry for the application PLU cannot be found.
- The contents of registers R14, R15, and R0 through R7 are unchanged, except as requested by the macro call.

## Programming Considerations

- This macro is for use by systems programs only.
- At assembly time the macro call is checked for:
  - A STEP parameter that is not valid or is missing
  - A ARG parameter that is not valid
  - A IN1 or IN2 parameter that is not valid or is missing
  - A EOT or EOTR label that is not valid or is missing.
- The registers specified by RVT1, RVT2, and RID are checked for validity.

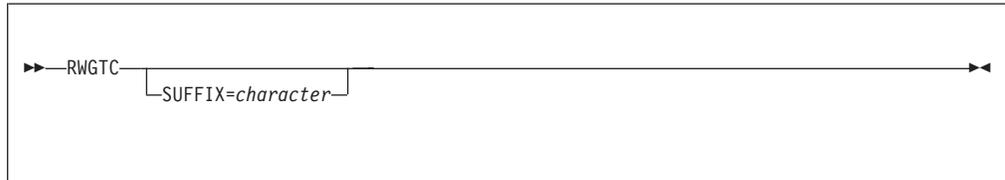
## Examples

None.

### RWGTC—Release a Lock on a WGTA Entry

Use this system macro to release a lock on an entry in the terminal address table (WGTA) that was locked previously by the LWGTC macro.

#### Format



#### **SUFFIX=character**

The argument specifies the suffix to be used for WGTA addressing. This is an optional parameter.

#### Entry Requirements

R9 must contain the address of the ECB being processed.

#### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Upon return, the WGTA entry is released.

#### Programming Considerations

- This macro can be run on any I-stream.
- The RWGTC must be issued on the same I-stream as the corresponding LWGTC.
- Addressability to the WGTA item must have been established prior to the use of this macro.
- The name of the WGTA lock byte is WG0LCK.
- The RWGTC macro must be issued in 31-bit mode since the WGTA can lie above 16 MB.

#### Examples

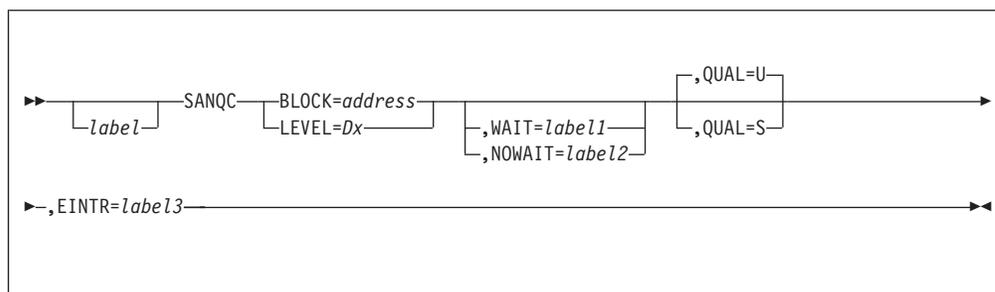
None.

## SANQC—Define and Enqueue Resource, Signal Aware

Use this system macro to define a shared resource to the control program and to control access to the resource among entry control blocks (ECBs). The SANQC macro is similar to the ENQC macro. However, if the caller of the SANQC macro has to wait for access to the shared resource, the caller can be interrupted by a signal. This macro is used with the DEQC macro.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=address**

If specified, this is the address of an 8-byte area that contains the name of the resource. BLOCK and LEVEL are mutually exclusive.

#### **LEVEL=Dx**

If specified, this is a file address reference word (D0–DF) that contains the name of the resource. BLOCK and LEVEL are mutually exclusive.

#### **WAIT=label1**

The label of an instruction that is given control if the ECB waited before gaining control of the resource. Otherwise, SANQC macro processing returns control to the next sequential instruction (NSI). WAIT and NOWAIT are mutually exclusive.

#### **NOWAIT=label2**

The label of an instruction to be given control if the named resource is already in use by another ECB. If specified and the resource is already in use, control is passed to the specified label without the ECB giving up control. NOWAIT and WAIT are mutually exclusive.

#### **QUAL**

Qualification of the resource name. The default value is U.

**U** Subsystem qualification applies. The resource name is subsystem unique and is qualified by the database index (DBI) value for the subsystem.

**S** Systemwide qualification applies. The resource name is not subsystem unique; that is, all ECBs in the TPF system that issue a SANQC macro with the same resource name and QUAL=S coded are enqueued on the same resource. If two SANQC or ENQC macros are issued with the same resource name, but different QUAL values are coded, two different

## SANQC

resources are assumed to exist. The SANQC macro and its corresponding DEQC macro must have the same QUAL value coded.

### **EINTR=***label***3**

The label of an instruction to be given control if SANQC macro processing is interrupted by a signal. When control is given to the instruction specified by the EINTR keyword, the caller does not have access to the shared resource. Because the caller does not have access to the shared resource, a system error results if the DEQC macro is called.

## Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- When the LEVEL keyword is specified, the caller must store the 1–8 character symbolic name of the shared resource in file address reference word CE1F*x*, where *x* is a data level (0–F).
- When BLOCK is specified, the caller must store the 1–8 character symbolic name of the shared resource in the 8-byte area pointed to by the address specified for the BLOCK keyword.

## Return Conditions

- If neither the WAIT or NOWAIT keyword is specified, control is returned to the next sequential instruction (NSI) when the ECB owns the resource.
- If the resource is already in use and the NOWAIT keyword is specified, control is given to the label specified by the NOWAIT keyword without the ECB losing control, but the ECB does not own the resource.
- If the resource is already in use and the WAIT keyword is specified, control is given to the label specified by the WAIT keyword when the ECB owns the resource.
- If the calling process receives a signal before the resource becomes available, control is given to the label specified by the EINTR keyword, but the ECB does not own the resource.
- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.

## Programming Considerations

- This macro can run on any I-stream.
- If the LEVEL keyword is specified, the ECB must not hold a block on the data level. If a block is held, a system error occurs and the ECB exits.
- When the resource is no longer needed, the ECB must issue a DEQC macro. If an ECB exits while holding a resource, a system error occurs and the resource is freed. See *TPF General Macros* for information about the DEQC macro.
- There is no timeout feature available on the SANQC macro as there is on the ENQC macro. However, the calling process can use the `alarm` function to specify a time period to wait for the resource to become available. If the resource does not become available in this time period, SANQC macro processing is interrupted by a SIGALRM signal.  
See *TPF General Macros* for information about the ENQC macro. See the *TPF C/C++ Language Support User's Guide* for information about the `alarm` function.
- SANQC macro processing does not handle signals. When a signal interrupts SANQC macro processing, control is passed to the instruction specified by the EINTR keyword.

- Both the ENQC macro and the SANQC macro can be used by different processes to access the same shared resource.

## **Examples**

None.

---

## SENDC—Send Message to Terminal

Use this system macro to transmit the following types of messages:

- Messages to high-speed display and printer terminals
- Messages to 3270 local terminals
- Messages to another system through binary synchronous communication lines (BSC) or synchronous link control (SLC)
- ATA/IATA format telegrams
- Messages to terminals supported by the Network Extension Facility (NEF)/ALCI, AX.25, and XALCI.

High-speed display and printer terminals attached to their terminal interchange units may be connected to the TPF system in one of the following ways:

- The terminal interchange can be connected to a high-speed (ALC) line into a transmission control unit attached directly to a subchannel of the multiplexor channel. This is referred to as a directly attached terminal.
- The terminal interchange can be connected through a high level network (for example, SITA HLN) that is attached to the TPF system through a synchronous link (AI). This is referred to as an indirectly attached terminal.

In a TPF system with NEF, the terminal interchange can be connected to a high-speed (ALC) line that in turn is connected to a 3725 with the NEF2 PRPQ.

Messages to directly attached display terminals and short reply messages to directly attached printer terminals are sent to the communication control program (CP) for code translation and initiation of output transmission.

Messages to indirectly attached display terminals, short reply messages to indirectly attached printer terminals and messages to another system through a synchronous link (AI) are sent through an intermediate queue mechanism to the TPF synchronous link control output interface programs.

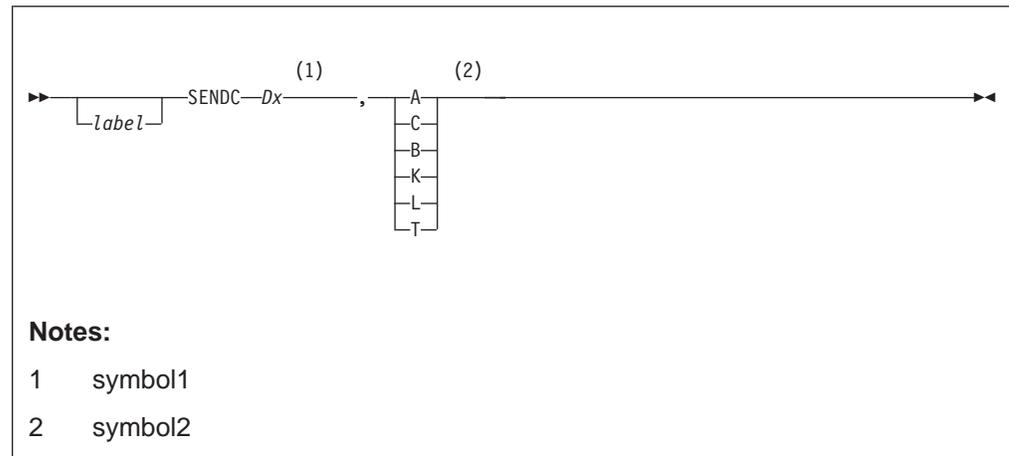
Long and unsolicited messages to printer terminals (directly or indirectly attached) are passed by the control transfer mechanism to the long message transmitter program.

ATA/IATA format telegrams are passed by the control transfer mechanism to the message switching system.

**Note:** The SENDC macros are restricted to system program use. Applications should use the ROUTC macro to transmit all messages. Any SENDC macro (class A, C, L, or B) issued by an application program is intercepted and converted to a ROUTC macro. The control program (CP) sets the CE1ACVT bit in the entry control block(ECB) control byte CE1CPA to indicate the conversion. When the control program (CP) issues a SENDC macro, the CE1AINT bit in CE1CPA must be set if the SENDC intercept routine (CCPNUC) is to be bypassed.

See *TPF General Macros* for more information about the ROUTC macro.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *symbol1*

Data level (D0 through DF) containing message segment to be sent.

### *symbol2*

A message class, one of the following:

#### **A or C**

Display Terminal reply messages or short Printer Terminal reply messages.

**B** Messages to another system through a binary synchronous (BSC) line.

**K** Messages to another system through a synchronous link.

**L** Long reply or unsolicited messages to printer terminals.

**T** ATA/IATA format telegrams

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The message segment must be contained in a block of storage held by the ECB at the specified level (*symbol1*).
- The maximum number of bytes in the segment depends on the message class. For class A, C, L, or B messages to terminals, only 381-byte file storage blocks are allowed.
- The format of a message segment follows.

No. of Bytes	4	4	4	4	2	
	A	B	C	D	E	F

Where:

**A** Contains record ID, RCC and control byte.

**B** Is the program ID field.

**C** Is the forward chain field.

**D** Is the backward chain field.

**E** Is the count of characters in Section F.

## SENDC

**F** The contents of this section vary according to the class of message.

### A,C,L

Terminal Address.

Text. See the UI0OM data macro for the OMSG format.

**B** Symbolic Line Number (SLN).

Symbolic Station Number

Three reserved bytes (bytes 20 through 22).

Text of the data for transmission over the BSC line. See the AM0SG data macro for the AMSG format.

**K** Symbolic Link Number (SKN).

Text. See the XM0RL data macro for the XMSG format. The format is defined by the XM0RL data macro only in so far as the first byte following the SKN is the text.

**T** Line Number-not used. The routing information is contained in the text.

Text. See the XM0RL data macro.

- If an application issues its own SENDC A, C, or L (bypassing the router and send intercept routine) byte B+2 of the message must be initialized with the terminal type indicator as defined in the TRMEQ macro.
- Class C-requires that the agents assembly area (the WA0AA data macro) be attached to the entry control block (ECB) on level 1.
- Class K-messages require additional information as follows:

### **1st Status Byte in Byte D**

Bit 0	0	Low Integrity Message. The message is not stored on file and cannot be regenerated in case of transmission errors.
	1	High Integrity. The message is stored on file and can be regenerated in case of transmission errors.
Bit 1	0	Reference is the F.A. of the message itself.
	1	Reference is the F.A. of an assembly area.
Bit 2	0	Type A message-short transit time required, for example, conversational messages.
	1	Type B message-longer transit time.
Bit 3	0	Assembly area is an XLMA.
	1	Assembly area is an AAA or RCB.
Bit 4	0	Do not release file address after transmission.
	1	Control program (CP) will release the message file address after successful transmission.
Bit 5,6,7		Defines transmission code across the link. 000 Padded CCITT No. 2 (Padded Baudot) 010 Padded 6 bit code (Padded Sabre) 100 CCITT No. 5 (ASCII/ISO, 7 bit) 110 Extended CCITT No. 5

### **2nd Status Byte in Byte D+1**

If the link connects to a high level network (HLN), additional routing information is required.

Bit 0		Reserved, must be set to 0.
Bit 1		Reserved, must be set to 0.
Bit 2	0	Normal message block chaining.
	1	Intermediate Prime Block. This is a technique intended for use by the message router package.
Bit 3	0	No rerouting of this message is required.
	1	Rerouting required.
Bit 4	0	Message to another system.
	1	Message to an indirectly coupled terminal.
Bit 5	0	Existing message format.
	1	AMSG format message.
Bit 6		Reserved.
Bit 7		Reserved.

### ***Byte B+2-Hex 2-byte address of Exit Center***

If the message is high integrity then the file address reference word of the ECB level to which the prime message block is attached must contain the file address of that block or an assembly area.

For class B messages, the Record ID, RCC and control byte in the block are used to retrieve and dispose of any message segments chained on file. The low-order three bits of the control byte (byte 3) are used as indicators.

Bit 5	0	For normal messages.
	1	If the message is an online test message.
Bit 6	0	for normal messages.
	1	If this is a priority message such messages are placed at the front of the line queue. The use of this bit should be restricted to system control messages.
Bit 7	0	If file storage blocks (overflow message segments) should be released after the message is sent.
	1	If file storage blocks should not be released after the message is sent.

- If a message is destined for a 3270 device, the high-order bit of the character count field must be set to one. This directs the CCP translation process.

## **Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The specified CBRW is initialized to indicate that a block of storage is no longer held.
- For class K messages, bit 3 of the second status byte is reserved for use by the message router package. If bit 3 of the second status byte = 1, then R15

## SENDC

contains a return code: 0 = line is operative, 4 = line is inoperative (in which case the core block specified is still attached to the appropriate level). In addition, if the link subsequently becomes inoperative, any link A-type messages (first status byte, bit 2=0) that have not been transmitted will be returned to the message router package.

## Programming Considerations

- Except for SENDC message class K, this macro can be run on any I-stream.
- SENDC K can be processed on the main I-stream only.
- Class A, C, L, or B messages are intercepted and converted to ROUTCs for processing by the message router (ROUT). If the calling ECB is in a commit scope, a system error is issued because the ROUTC macro is not supported by TPF transaction services.
- Class A or C messages to display terminals may exceed the capacity of one message segment. In this case, the macro must be issued once for each block of core comprising the complete logical message. No WAITCs are allowed between successive SENDCs for the same logical message.
- Class B, K, L, or T messages may be any length in which case only the first message segment is in main storage and presented to the control program (CP) by the SENDC macro; the remaining message segments are on file, chained to the prime segment in the standard way.
- Class K messages that overflow to file must contain in the prime block a character count, greater than LK4MXT (field in LK4KC link keypoint which defines the maximum text allowed in each link information block) +1 (for the one byte SLN character).
- A check is made by the control program (CP) to determine if the ECB is holding a block of storage at the specified level. If not, control is transferred to the operational system error routine and the ECB exited.
- For Class A, B, C and L messages the line number in the message block is checked to ensure that it is within the range of existing lines in the TPF system. If not an existing line, it is checked to see if it is a pseudo line corresponding to a terminal supported by NEF/ALCI, AX.25 or XALCI. If not, a system error is issued and the ECB is exited.
- For Class K messages, the Link identification is checked to ensure that it is within the range of existing links in the TPF system. If it is not, control is transferred to the operational system error routine and the ECB is exited.
- The Send routine checks that the specified line is usable. If not, a system error is generated, the output operation is ignored and a normal macro return is used. If the line is usable the last character is checked for an EOM character. If it is not, control is transferred to the operational system error routine and the ECB exited. Messages destined for NEF/ALCI, AX.25, and XALCI pseudo lines are processed by ALC via the SNA output interface. This routine verifies that the SNA link supporting this terminal is available. If not, a system error is taken and the ECB is exited.
- The status of the sending operation can never be determined by the operational program.
- The operational program may use the specified level immediately upon return from this macro.
- Unsolicited messages to RO or Prime CRAS should contain LN,IA of X'0000' or X'0100' respectively and be sent using Class L.
- Class L messages are passed to program XLMT.
- Class T messages are passed to program XHAP.

- Class K messages cause program CMS to be activated.
- If a class B message cannot be sent, it is given to the Router alternate line selector (RALS).

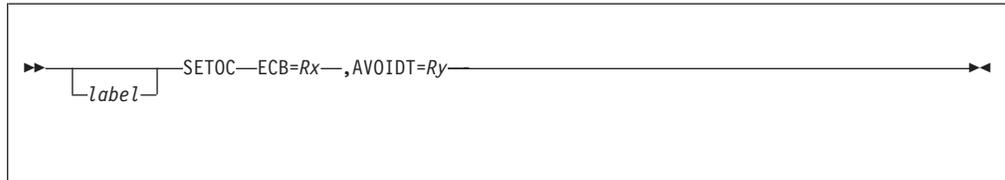
## Examples

None.

## SETOC—Set Maximum Times to Avoid Application Timeout for an ECB

Use this system macro to set the maximum times that an application can avoid a timeout between loss of control for the specified entry control block (ECB).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ECB=Rx**

The system virtual address (SVA) of the target ECB that is to avoid application timeout. The SVA is specified in a register in the range R0 through R6.

#### **AVOIDT=Ry**

The maximum times the application is to avoid timeout. This is in addition to the one time for the standard system timeout. The value is specified in a register in the range R0 through R6.

The value acts as a multiplier of the standard system timeout value.

Specify a value of 0-32766.

A value of 0 means the ECB will use the standard application timeout.

### Entry Requirements

- C-type programs must be in key 0 and supervisor state when processing this macro, and R13 must point to a valid system stack area.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Return codes in R15:
  - If no error occurs, R15 contains zero, and the timeout for this ECB is changed.
  - If an error occurs, an error indication code is returned in R15, and the timeout is not changed. Control is returned to the caller after the first error is found.
- For E-type programs, the contents of R10 and R14 are unknown.
- The contents of all other registers are preserved across this macro call.
- The following tags are generated by the SETOC macro and should be used for interrogating the error indication code returned in R15.

**IT\_SETOC\_BAD\_ECB\_ERR** The address in the ECB parameter register is not a valid system virtual address (SVA) ECB address. The timeout for the ECB is not changed.

**IT\_SETOC\_BAD\_AVOIDT\_ERR** The value in the AVOIDT parameter register is a

negative number or a number greater than 32766. The timeout for the ECB is not changed.

## Programming Considerations

- The contents of the AVOIDT parameter register acts as a multiplier of the standard system timeout value.  
For example, if the standard system timeout value is 500 milliseconds, a value of 3 causes the ECB to wait 3 times in addition to the 1 time for the standard system application timeout. The ECB will wait ((3 times 500) plus (1 times 500)), or 2000 milliseconds, before timing out.
- Specify the minimum AVOIDT value needed. A large value can cause system performance problems or lockout problems.
- The ECB issuing the SETOC macro must be on the same I-stream as the target ECB, or the ECB should pause the TPF system before issuing the SETOC macro on behalf of another ECB.
- You can use the ZCTKA command to change the value in keypoint A for the number of times to avoid an application timeout. TPF system code can use the contents of the CINFC CMMAVMAX field for the contents of the SETOC macro AVOIDT parameter register when issuing the SETOC macro.
- The SETOC macro does not affect the CTL-2010 time slice timeout.
- The APL user exit is provided for you to issue the SETOC macro to set the maximum times to avoid timeout for an ECB right before the ECB is about to be timed out with a CTL-10.
- The C function trace CDEB, CEXP, and CTRC user exits are provided for you to issue the SETOC macro to set the maximum timeout value for ISO-C programs that have been compiled using the TEST compiler option of one of the IBM C/370 family of compilers supported by the TPF 4.1 system.
- The SETOC macro sets R15 with a return code. If the SETOC macro is used in the CDEB, CEXP, CTRC, or APL user exits, a base register other than R15 should be established for addressability.

## Examples

In the following examples, the standard application timeout value is assumed to be 500 milliseconds. For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- AVOIDT parameter register contents of 2:  
(R6 contains the SVM address of an active ECB in the TPF system)  
LA R3,2  
SETOC ECB=R6,AVOIDT=R3

This invocation:

- Changes the timeout for the ECB whose SVA is contained in register 6. The ECB will wait 2 times the standard timeout in addition to the 1 time for the standard timeout. The number of milliseconds the ECB will wait before timing out is  
- ((2 times 500) plus (1 times 500)), or 1500 milliseconds.
- AVOIDT parameter register contents of 32767:  
(R6 contains the SVM address of an active ECB in the TPF system)  
LA R3,=X'00007FFF' VALUE IS 32767  
SETOC ECB=R6,AVOIDT=R3

## SETOC

This invocation causes an error condition to be returned in R15 because an AVOIDT value greater than 32766 is an error condition. The timeout is not changed.

- AVOIDT parameter register contents of 0:  
(R6 contains the SVM address of an active ECB in the TPF system)  
LA R3,0  
SETOC ECB=R6,AVOIDT=R3

This invocation resets the timeout to the standard application timeout for the ECB whose SVA is contained in register 6. If the standard application timeout is 500 milliseconds, the ECB will wait 500 milliseconds before timing out.

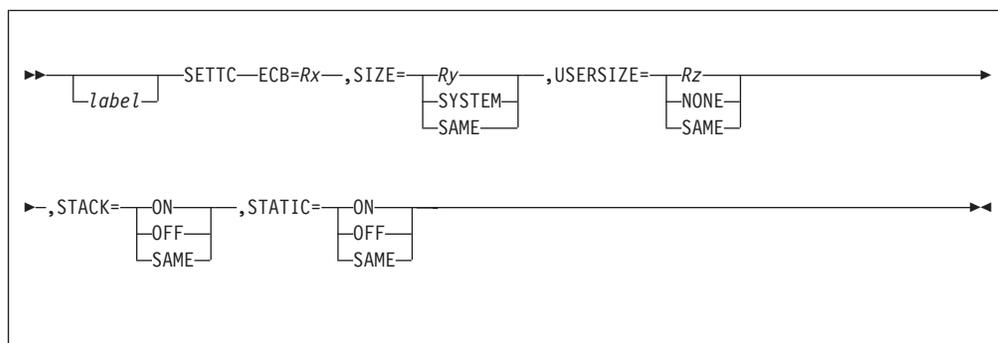
## SETTC–Set C Function Trace Information for an ECB

Use this system macro to change C function trace information for the specified entry control block (ECB). By using this macro you can:

- Specify a particular ECB to change
- Specify the size of the C function trace table to allocate
- Specify the size of the C function trace user area to allocate
- Change the subtrace of stack data.
- Change the subtrace of static data.

After a C function trace table has been allocated for an ECB, this macro has no effect, except that an error indication code is returned in register 15 (R15).

### Format



#### *label*

A symbolic name assigned to the macro statement.

#### **ECB=Rx**

Changes the C function trace settings for the ECB with its system virtual address (SVA) specified in a register in the range R0 – R6.

#### **SIZE**

Specifies the size of the C function trace table to allocate for the specified ECB. The size is saved and used later when the C function trace table is created.

If a C function trace table already exists for the specified ECB, the value specified by the SIZE parameter is not saved and an error indication code is returned in register R15.

*Ry* A register in the range R0 – R6 containing the number of 4096-byte (4 KB) blocks of heap storage to allocate for the C function trace table. This number is from 1 to a maximum of 256 (X'100').

If the value in the register does not fall within the 1 – 256 range, the value is not saved and an error indication code is returned in register R15.

#### **SYSTEM**

The TPF system value for the trace table size is one 4 KB block (4096 bytes).

#### **SAME**

Do not change the size of the C function trace table. If the size has not been specified by a previous SETTC macro, a value of one 4 KB block (4096 bytes) is used when the C function trace table is created.

## SETTC

### USERSIZE

Specifies the size of the C function trace user area to allocate when tracing starts for the specified ECB.

If a C function trace user area already exists for the specified ECB, the value specified by the USERSIZE parameter is not saved and an error indication code is returned in register R15.

*Rz* A register in the range R0 – R6 containing the number of 4096-byte blocks of heap storage to allocate for the C function trace user area. This number is in the 1 to 256 (X'100') range.

If the value in the register does not fall in the 1 to 256 range, the value is not saved and an error indication code is returned in register R15.

### NONE

Do not allocate a C function trace user trace table.

### SAME

Do not change the existing size of the C function trace user area. If the size has not been specified by a previous SETTC macro, a C function trace user area will not be created.

### STACK

Changes tracing of C stack data for the specified ECB.

#### ON

Specifies that 68 bytes of the stack user information are placed into the trace table entry.

#### OFF

Specifies that the address of the stack area is placed into the trace entry and no C stack information is placed in the trace table.

#### SAME

Do not change the current setting of C function trace stack data. If the setting of STACK has not been specified by a previous SETTC macro, the current system STACK value is used.

### STATIC

Changes tracing of C static data for the specified ECB.

#### ON

Specifies that 68 bytes of the C static information are placed into the trace table.

#### OFF

Specifies that the address of the static area is placed into the trace entry, but no C static data is placed in the trace table.

#### SAME

Do not change the current setting of C function trace of static data. If the setting of STATIC has not been specified by a previous SETTC macro, the current system STATIC value is used.

## Entry Requirements

- For C-type code, the code must be in key of zero and supervisor state when executing this macro, and R13 must point to a valid system stack area.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- The code issuing the SETTC macro must be on the same I-stream as the target ECB.
- The TPF system should be paused before issuing this macro on behalf of another ECB.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- For C-type code, register R15 contains the return code, and the contents of all other registers are preserved across this macro call.
- For E-type code, the contents of R10 and R14 are unknown, R15 contains the return code and the contents of all other registers are preserved across this macro call.
- If no errors occur, register R15 contains zero.
- If an error occurs, no C function trace information is saved and an error indication code is returned in register R15. If there are multiple errors, control is returned to the caller after the first error is found.
- An error indication code is returned in register R15 when:
  - The ECB address in the register specified by the ECB parameter is not a valid system virtual address (SVA) ECB address.
  - The C function trace table or the user area already exists for the specified ECB.
  - A register is used with the SIZE parameter and the value in the register does not fall in the 1 to 256 range.
  - A register is used with the USERSIZE parameter and the value in the register does not fall in the 1 to 256 range.
- The following tags are generated by the SETTC macro and should be used for interrogating the error indication code.

IT_BAD_CID_ERR	Incorrect CID SVM address in target ECB detected by \$GSVAC macro.
IT_BAD_ECB_ERR	Not a valid system virtual address (SVA) ECB address.
IT_BAD_TCA_ERR	Incorrect TCA SVM address in target ECB detected by \$GSVAC macro.
IT_SIZE_ERR	The value in the SIZE register is not in the range of 1 to 256.
IT_TRACE_TABLE_ERR	C function trace table already exists.
IT_USER_AREA_ERR	C function trace user area already exists.
IT_USER_SIZE_ERR	The value in the USERSIZE register is not in the range of 1 to 256.

## Programming Considerations

- Once a C function trace table has been allocated for an ECB, the SETTC macro cannot change the C function trace settings and returns an error indication code in register R15.
- On any error, the SETTC macro does not change the C function trace settings and returns an error indication code in register R15.

## SETTC

- The C function trace user area is an optional storage area, unique per ECB, the contents of which are controlled by the user. Its purpose is to provide a user area to store additional information beyond what is stored in the C function trace table.

## Examples

- This invocation:

```
(R4 contains the SVM address of an active ECB in the TPF system)
LA    R5,11
LA    R6,3
SETTC ECB=R4,SIZE=R5,USERSIZE=R6,STATIC=ON,STACK=OFF
```

- Changes the C function trace information for the ECB whose SVA is specified in register 4.
- Specifies the size of 11 (45 056 bytes, or X'B000' bytes) for the C function trace table.
- Specifies the size of 3 (12 288 bytes, or X'3000' bytes) for the C function trace user area.
- Starts the sub-tracing of static data when C function trace is later activated through the ENATC macro or the ZSTRC command.
- Stops the sub-tracing of stack data (only the stack address is traced) when C function trace is later activated via the ENATC macro or the ZSTRC command.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- This invocation:

```
LA    R5,4
SETTC ECB=R6,SIZE=R5,USERSIZE=NONE,STATIC=ON,STACK=ON
```

- Changes the C function trace information for the ECB whose SVA is specified in register 6.
- Specifies the size of 4 (16 384 bytes, or X'4000' bytes) for the C function trace table.
- Specifies that there should be no C function trace user area.
- Starts the sub-tracing of static and stack data when C tracing is later activated via the ENATC macro or the ZSTRC operator command.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

- This invocation:

```
(R4 contains the SVM address of an active ECB in the TPF system)
LA    R6,8192
SETTC ECB=R4,SIZE=R6,USERSIZE=SAME,STATIC=ON,STACK=OFF
```

- Does not save any C function trace information.
- Returns the error indication code IT\_SIZE\_ERR in register R15.  
Reason: 8192 is greater than the maximum value of 256 (X'100'). The specified value should be the number of 4096-byte blocks to allocate.  
If the caller wanted to specify a C function trace table size of 8192 (X'2000') bytes, the caller should load register R6 with the value 2.
- For E-type programs, R9 must contain the ECB virtual address (EVA) of the ECB issuing the macro.

## SICFC–IPC Service Request

Use this system macro to request special system inter-processor communication function services.

### Format

```
▶▶—SICFC—RTYPE=FLUSHQ—,REQ=Rx————▶▶
```

#### RTYPE

This required parameter specifies the type of request you are making. The only valid request is:

#### FLUSHQ

This service will send all SIPC items that have been queued for a specific processor.

#### REG=R<sub>x</sub>

This parameter specifies a register containing the address of the entry for the specific processor in the IPC global table (IGT). Specify a general register in the range R0 through R7. The IGT is defined by the DCTIGT DSECT.

### Entry Requirements

R14 and R15 must be available for use by this macro.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R0 through R13 are preserved across this macro call.

### Programming Considerations

- This macro can only be run on the main I-stream only.
- The SICFC macro has an interface that is not guaranteed across releases of the TPF system. Unauthorized use of this macro may expose you to interface and/or processing errors.
- This macro is intended to be used by system IPC support programs only.

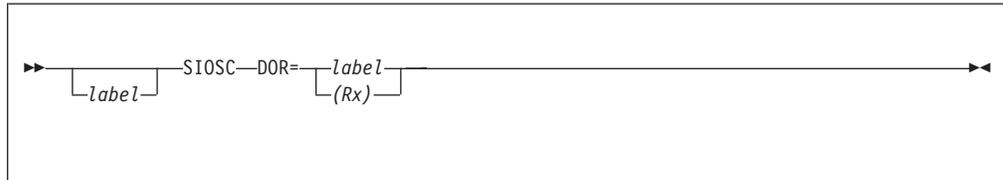
### Examples

None.

## SIOSC—Start an Input/Output (I/O) Operation

Use this system macro to start a normal input/output (I/O) operation to a specific symbolic device address (SDA).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **DOR=label(Rx)**

This is either a label assigned to an area defined by the device operation request block (DCTDOR) DSECT or a register containing the address of a DOR.

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW Key 0.
- The DCTDOR DSECT completed with:
  - SDA address
  - Protect key for data area
  - CCW format flag
  - Logical path mask
  - Channel program address.
- And optionally set:
  - Lost interrupt timeout value
  - Request-unique parameter.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The condition code will be set as follows.
 

Condition Code	Meaning
0	The request was accepted.
2	A request was already active.
3	The SDA is not valid.
- The contents of all registers are preserved across this macro call.

### Programming Considerations

This macro is for use in the control program (CP) only.

### Examples

None.

## SIPCC—System Interprocessor/Inter-I-Stream Communication

Use this system macro to provide communication among processors in a TPF loosely coupled environment and among instruction streams in a tightly coupled environment.

This macro causes an interprocessor communication (IPC) item to be transmitted to a specified program segment in one or more active I-streams, in one or more active processors. The destination processors can be targeted individually, in various combinations or in a broadcast request. The destination I-streams can be targeted as the main I-stream or all I-streams in the destination processors.

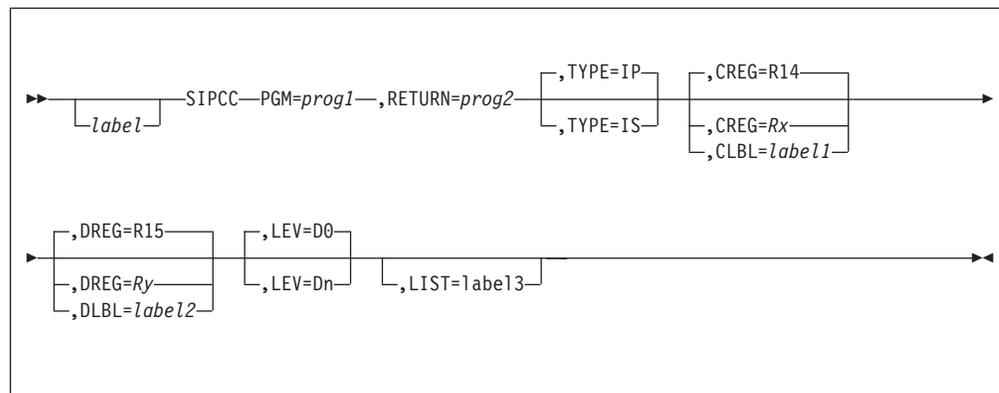
An IPC item contains the following information:

- An 8-byte user control area
- A 24-byte IPC control area
- 2 variable length data areas (this is optional).

Data area 1 (DA1) can be no more than 104 bytes while data area 2 (DA2) can be any length that fits in a single 128, 381, 1055, or 4 K byte storage block.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

**PGM=prog1**

This is a 4-character name identifying the program segment invoked in the receiving processors or I-streams for which this item is intended.

**RETURN=prog2**

This is a 4-character name identifying the program segment in this processor to be given control for each active destination processor that reads or fails to read this processor's transmission.

**TYPE**

This parameter is used to specify the type of processing requested.

**IP** Indicates interprocessor communication is requested and will transmit the

## SIPCC

SIPCC request to the specified processors. The request will not be transmitted to the originating processor. This is the default.

**IS** Indicates inter-I-stream communication is requested and will transmit the SIPCC request to the main I-stream or all I-streams on the specified processors including the originating processor. The request will not be transmitted to the originating I-stream.

**CLBL=*label1***

This is a symbolic name identifying the location of an 8-byte control area.

**CREG=R14|*Rx***

This is a symbolic name identifying the register that contains the address of an 8-byte control area.

*Default:* If neither CLBL nor CREG is specified, register R14 is assumed to have a pointer to the control area.

**DLBL=*label2***

This is a symbolic name identifying the location of a variable length data area (DA1).

**DREG=R15|*Ry***

This is a symbolic name identifying the register that contains the address of a variable length data area (DA1).

*Default:* If neither DLBL nor DREG is specified and the SI3LEN1 field in the control area (SI3CT) is not zero, register R15 is assumed to have a pointer to data area 1.

**LEV=D0|...|DF**

This is a symbolic value, D0-DF, which identifies the ECB level containing the storage block to be transmitted (DA2).

*Default:* If LEV is not specified and the SI3LEN2 field in the control bytes is not zero, a level of zero is assumed.

**LIST=*label3***

This is a symbolic name that identifies the name of a variable list of destination processors.

The LIST parameter can be specified with all destination types. If specified, the SIPCC macro service routine returns a list of all processors to which transmission was started.

## Entry Requirements

- R9 must contain the address of the entry control block (ECB) being processed.
- You must set up an 8-byte control area. Optionally, you can set up one or two data areas and a list of destination processors. The setup includes initializing the destination, flag, and length fields in the control area, as well as establishing the data areas.
- The address of the control area must be put in R14 or be specified with the CLBL or CREG parameter.
- The address of the optional first data area, DA1, must be put in R15 or specified with the DLBL or DREG parameter.
- The address of the optional second data area, DA2, is specified in the storage block reference word of the ECB at the level specified by the LEV parameter. DA2 must reside in a storage block and LEV must be specified if the length field for DA2 is not zero. If DA2 exists, the data transmitted will begin at byte 0 of the storage block for the specified length.

- Processor destination information is specified in the IPC control area (SI3CT). Destination types are specified by setting the destination field (SI3DEST) to the following values:
  - Broadcast destination (SI3DEST = X'FF')
  - Destination processor list (SI3DEST = X'FE')

**Note:** The list of destination processors is specified by the LIST parameter. To generate a list of destination processors, use the GENLC macro. See *TPF General Macros* for information about the GENLC macro.

- Specific destination processor (SI3DEST = X'nn', where *nn* is the processor ordinal number).

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 and the condition code are unknown. The contents of all other registers are preserved across this macro call.
- The activity field (SI3ACT) in the control area is set to the number of active destination processors for which the transmission was started. The system interprocessor communications facility (SICF) uses the Processor Status Table (PI1DT) to determine processor activity. If there are no other active processors when SIPCC is requested, SI3ACT is set to zero.

If the destination was selective and the LIST parameter was specified, the returned list contains the ordinal numbers of the destination processors for which the transmission was started.

**Note:** The original contents of the list are not preserved. The list is overlaid by the SIPCC macro service routine on return to the user.

If the destination was broadcast (sent to all active processors) and the LIST parameter was specified, the returned list contains the ordinal numbers of the destination processors for which the transmission was started.

The destination field (SI3DEST) in the user control area is unchanged.

- A flag (SI3INAC) is turned on in the control area if any destination processor specified in SI3DEST is not active when SIPCC macro processing begins.
- The storage block provided for data area 2 will remain attached to the ECB upon return of control at NSI.
- SICF will invoke the program segment specified by the RETURN parameter for each destination processor that:
  - Reads this processor's transmission if the SI3CT flag SI3XMIT is set (=1)
  - Fails to read this processor's transmission if the SI3CT flag SI3XERR is set (=1).

The control fields and data areas will be set up by SICF in the same format as for a receiving program. The SI3RETD flag identifies the invocation as a RETURN. The SI3RERR flag will be off (=0) if the transmission completed successfully and the SI3XMIT flag was set (=1). SI3RERR will be on (=1) if the transmission failed to be completed and SI3XERR was set (=1). SI3PROC contains the ordinal number of the destination processor that read or failed to read the transmission.

## SIPCC

**Note:** Special considerations exist when RETURN and TYPE=IS are specified. I-streams on the originating processor may receive transmissions but, since the originating processor is not considered an IPC destination, will not invoke a program RETURN.

## Programming Considerations

- This macro may be processed on any I-stream.
- The SIPCC macro can be issued only by a E-type program.
- The ECB reference register (R9) must contain the address of the ECB before issuing the SIPCC macro.
- ECBs created by the SIPCC macro use the version of the program most recently activated on the specified processor. If this program is incompatible with the program that issued the SIPCC call, an interface problem may occur.
- The CREG and DREG parameters must be coded or defaulted to different registers.
- The user of SIPCC can be notified of successful and/or unsuccessful transmissions, for each active destination processor, by coding the macro RETURN parameter and setting the SI3XMIT and/or SI3XERR flags in the user control area (SI3CT). See the previous note for TYPE=IS considerations.
- When using the I-stream routing request the user will specify the I-stream destination by setting the SI3ALL\_IS flag in the user flag byte (SI3FLGU). When routing to the main I-stream set SI3ALL\_IS to 0. When routing to all I-streams set SI3ALL\_IS to 1. Requests will not be routed to the originating I-stream.
- No attempt is made to transmit to an inactive processor. If the request was a broadcast to all processors in the complex and any processor is not active, flag SI3INACT is set (=1).
- IPC items that require expedited handling are identified by setting the priority flag (SI3PRTY) in the control area to 1. Priority requests cause incore staging to be halted and transmission for all available items to be initiated. In the receiving processors, priority items are placed on the ready list.
- The program segment named in the PGM and RETURN parameters must reside on the data base indicated by the PBI field in the requestor's ECB. That is, the program must have been allocated using the system allocator and loaded to the system files. See *TPF System Installation Support Reference* for more information about the system allocator.
- The PBI and SSUID fields in the requestors ECB will be transmitted to the destinations. There, they will be used to initialize the ECB and invoke the specified program.
- The format of an IPC item as seen by the destination program segment and the program segment specified in the RETURN parameter is as follows:
  - The 8-byte control area will be placed in the second work area of the ECB beginning at location EBX000.
  - Data area 1 will be placed in the first work area of the ECB beginning at location EBW000.
  - Data area 2 will be placed in a storage block on level 0 of the ECB. The size of the storage block will be the same as the sender's block size.
- In addition to the normal macro trace information, the SIPCC macro trace entry will contain the destination processor indicator, the origin I-stream, the system flag field, and the user flag field.

## Examples

- Broadcast data from location EBW008 to segment CDEF in all active processors. The control area is defined at location EBW000.

```
SIPCC PGM=CDEF,CLBL=EBW000,DLBL=EBW008
```

**Note:** Since the control area will be modified by SICF, it must reside in modifiable storage.

- Same as previously except that the address of the control area has been placed in register 2.

```
SIPCC PGM=CDEF,CREG=R2,DLBL=EBW008
```

**Note:** SICF will load R15 with the address of EBW008. R15 could not, therefore, have been used for CREG.

- Broadcast data from location EBW008 to segment CDEF in all active I-streams in all active processors.

```
SIPCC PGM=CDEF,TYPE=IS,CREG=R2,DLBL=EBW008
```

For this example the user has set SI3ALL\_IS to '1'. Also, SI3DEST can be set to 'FF' to indicate all processors.

- The data being transmitted has now been moved to a storage block and its address loaded in register R15. The user wishes to have this data appear in the destination segment's ECB.

```
SIPCC PGM=CDEF,CREG=R2
```

**Note:** Neither DREG nor DLBL had to be specified since R15, the default register, contains the address of the data.

- In this example, the data is in the same storage block (on level 4) but you want the destination segment to receive it in an equivalent storage block.

```
SIPCC PGM=CDEF,CREG=R2,LEV=D4
```

**Note:** Destination segment CDEF will receive the storage block on level 0.

- Having preloaded R14 and R15 with the control area and data area addresses, you may want to notify the KLMN segment of successful transmissions to each of the destination processors.

```
SIPCC PGM=CDEF,RETURN=KLMN
```

**Note:** The SI3XMIT control area flag had to be set before issuing the SIPCC macro.

- You may want a response from the destination segment. There are to be no data areas transmitted.

```
SIPCC PGM=CDEF
```

**Note:** The SI3LEN1 control area field must be zero otherwise SICF will assume R15 contains the address of a data area. You must also have preloaded R14 with the address of the control area. The SIEREQR control area flag may be used to request a response. SICF does not make use of this flag.

- You can specify a set of destination processors by using a data list constructed by the GENLC macro. For this, the SI3DEST control area field is set to 'FE' and the address of the list is passed to the SIPCC service routine by the LIST parameter.

```
SIPCC PGM=CDEF,CREG=R2,LEV=D4,LIST=EVNBKLW
```

## SIPCC

**Note:** On return, the contents of the list block are modified to hold a list of the processors to which transmission was started. The list block can be empty (list count of 0).

- When specifying a broadcast transmission (SI3DEST=X'FF'), you can have the SIPCC macro service routine return a list of processors to which the transmission was started. This returned list can be used with the Internal Event Facility (LIST type event).

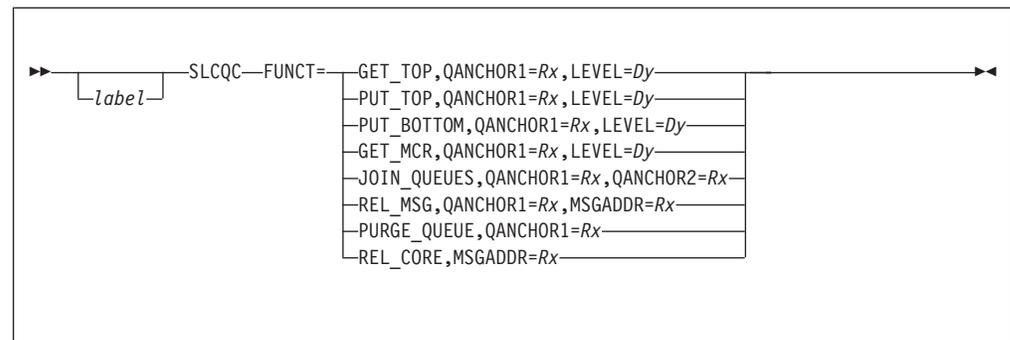
```
SIPCC PGM=CDEF,CREG=R2,LEV=D4,LIST=EBX000
```

## SLCQC–SLC Queue Handling

Use this system macro to have an entry control block (ECB)-controlled program request that the control program (CP) manipulate the synchronous link control (SLC) queues and release storage in the system virtual memory (SVM).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FUNCT**

Name of function to be performed. This required parameter must be one of the following values:

##### **GET\_TOP**

Retrieve a message from the top of the queue.

##### **PUT\_TOP**

Place a message at the top of the queue.

##### **PUT\_BOTTOM**

Place a message at the end of the queue.

##### **GET\_MCR**

Retrieve a queued message which qualifies for placement in the message control record (MCR).

##### **JOIN\_QUEUES**

Adds a channel queue to the front of a link queue.

##### **REL\_MSG**

Removes a message from the queue and releases the core block.

##### **PURGE\_QUEUE**

Purge the queue. R15 contains the number of messages purged.

##### **REL\_CORE**

Releases a core block residing in the SVM.

#### **QANCHOR1=Rx**

Specifies a register (R0 through R7) that contains the address of the anchor of the queue to be manipulated. It is required for all functions except REL\_CORE.

## SLCQC

### **LEVEL=***symbol*

Specifies a core block reference word (D0-DF). It must be specified for the following functions:

- GET\_TOP
- PUT\_TOP
- PUT\_BOTTOM
- GET\_MCR.

### **QANCHOR2=***Rx*

Specifies a register (R0 through R7) that contains the address of the anchor of the queue to be added to the top of the queue referenced by QANCHOR1. It is required for the JOIN\_QUEUES function.

### **MSGADDR=***Rx*

Specifies a register (R0 through R7) that contains the system virtual address (SVA) of a message to be released. It is required for REL\_MSG and REL\_CORE.

## Entry Requirements

- This macro must be called from the ECB virtual memory (EVM).
- R9 must contain the address of the program's ECB.

## Return Conditions

- Control is returned to the next sequential instruction NSI.
- The message that was dequeued is attached to the data level specified.
- R15 contains a return code as follows.

Return Code	Function	Meaning
0	GET_TOP GET_MCR REL_MSG	Message returned on data level specified.
4		Queue empty or message not found.
0	PUT_TOP PUT_BOTTOM JOIN_QUEUES REL_CORE	Processing complete.

## Programming Considerations

- This macro is restricted to SLC ECB-controlled program usage on the main I-stream only.
- The GET\_MCR function searches the queue for a message with the following characteristics:
  - High integrity
  - Not a duplicate
  - Not being retransmitted because of an earlier problem.

## Examples

```
MYLABEL SLCQC FUNCT=GET_TOP,QANCHOR1=R3,LEVEL=D0
```

The first message on the queue pointed to by the address in register R3 is detached from the queue (dequeued) and attached to ECB data level 0.

```
MYLABEL SLCQC FUNCT=PUT_TOP,QANCHOR1=R1,LEVEL=D3
```

The message attached to data level 3 is detached and added to the top of the queue pointed to by the address in register R1.

```
MYLABEL SLCQC FUNCT=JOIN_QUEUES,QANCHOR1=R1,QANCHOR2=R2
```

The channel queue pointed to by the address in R2 is placed at the beginning of the link queue pointed to by the address in register R1.

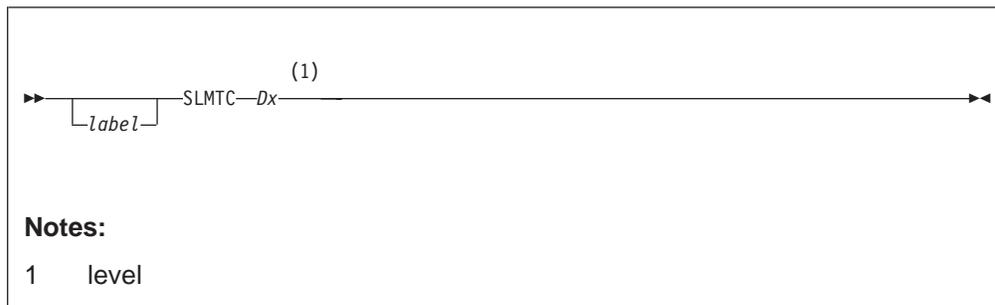
## SLMTC—Send LMT High Speed Transmission

Use this system macro to differentiate those messages that are destined for high-speed terminals from those messages that are destined for the console. Nonconsole messages are routed to the SENDA macro while console messages are sent through the console input/output (I/O) routines and a software answerback is generated to the long message transmitter (LMT).

This macro is used exclusively by the LMT for all its high-speed transmission, including transmission to a high-speed device supported by the Network Extension Facility (NEF).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

*level*

A core block reference word (D0-DF).

### Entry Requirements

- The message to be sent is contained in a block of storage held by the ECB at the specified level.
- The message to be sent contains the message header as described in the SEND macro, and is ended by the EOM character.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The specified core block reference word (CBRW) is initialized to indicate that a block of storage is no longer held.
- The contents of scratch registers R14 and R15 are unknown. The contents of the remaining operational registers and the condition code are saved during processing of this macro.

### Programming Considerations

- The ECB reference register (R9) contains the address of the ECB.

- A check is made by the control program (CP) to determine if the ECB is holding a block of storage at the specified level. If a block is not held, control is transferred to the operational system error routine.
- The block of storage containing the message to be sent is no longer available to the operational program.
- The status of the sending operation can never be determined by the operational program except through the use of an answerback timeout feature.
- Messages sent with the SLMTC macro are queued by LMT waiting an answerback acknowledgement. After a specified time passes without an answerback being received, a message is retransmitted. LMT retransmits 3 times before sending a reset message. The LMT timeout runs every 9 seconds or the number of intervals specified on an XS0OTM REP card in the XLMA/XLMT section of the STC input deck used to create the UAT/XLMA/XLMT records.
- The operational program may use the specified level immediately upon return from this macro.
- This macro can be run on the main I-stream only.

## Examples

None.

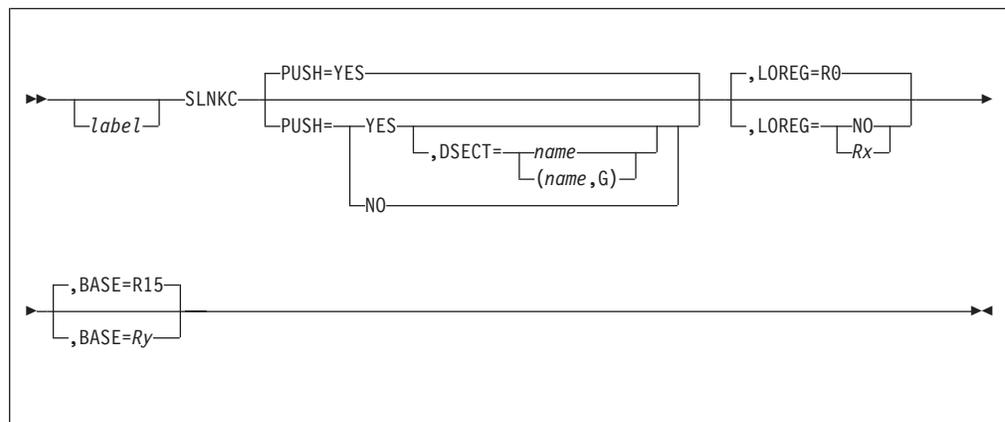
## SLNKC–Control Program (CP) Save Link Data & Set Stack Pointer

Use this system macro to save the caller's registers and, optionally, push the stack pointer to provide the called routine with a save area and a work area with its corresponding DSECT and USING statements.

This macro is used with other standardized linkage macros such as the CLNKC, DLNKC, and RLNKC macros. See the following for more information about these macros:

- "CLNKC–Control Program (CP) Call and Link" on page 166
- "DLNKC–Define Stack DSECT for Control Program (CP) Routine" on page 216
- "RLNKC–Return to CP Calling Routine and Reset Stack Pointer" on page 409.

### Format



*label*

A symbolic name can be assigned to the macro call.

#### **PUSH=YES|NO**

Specifies whether or not the stack pointer will be pushed. DSECT and PUSH=NO are mutually exclusive. YES is the default.

#### **DSECT=name|(name,G)**

The name of the DSECT to be used in mapping a work area. The name must be no longer than 4 characters. If DSECT=(name,G) is coded, then space will be allocated on the stack for the DSECT, but no DSECT or USING statement will be generated. This allows multiple SLNKC macros in a given assembly to refer to the same DSECT. Specification of DSECT is optional. DSECT and PUSH=NO are mutually exclusive.

#### **LOREG**

This is NO or the lowest numbered register that is altered by the routine issuing this SLNKC macro. NO indicates that no registers are to be saved. All the registers from the one specified to R15 (except for the register specified by the BASE parameter) are saved on the stack. A self-defining term cannot be used to specify a register. R0 is the default.

#### **BASE=R15|Ry**

This means the base register of the routine issuing this SLNKC macro. A self-defining term cannot be specified. R0 and R13 must not be specified. R15 is the default.

## Entry Requirements

- R13 must point to the appropriate stack.
- You must code this macro at the beginning of a called control program (CP) routine.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- If PUSH=YES, then R13 points to this routine's stack. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream, in a control program (CP) routine only. When DSECT=name is specified the DLNKC macro must also be coded.
- This macro is for use in the control program (CP) only.

## Examples

```
SLNKC DSECT=ABC,BASE=R10
```

In this example a USING statement and a DSECT named ABC will be generated, registers R0 through R9 and R11 through R15 will be saved, and R13 will be set to the appropriate stack address. The lines immediately following the invocation of SLNKC would typically look like the following.

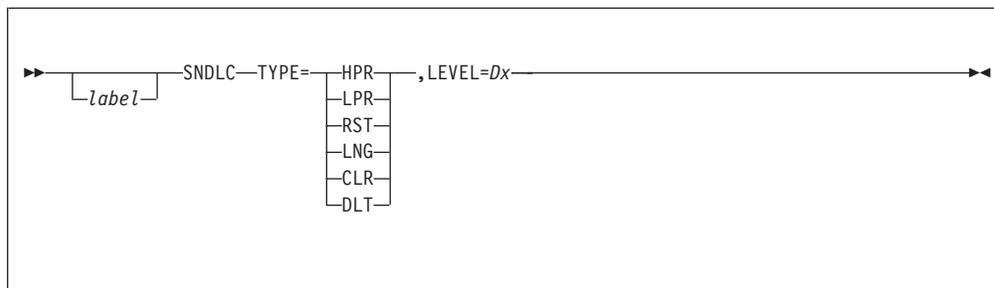
```
ABC1    DS    F        1st field of DSECT ABC
...
...
ABCn    DS    F        Last field of DSECT ABC
        DLNKC ,      End of DSECT ABC, resumption of CSECT
```

## SNDLC—Send Control Message to 3270

Use this system macro to allow the communication control program (CCP) to send control or multiple segment core-chained messages to a 3270 local device.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name may be assigned to the macro statement.

**TYPE**

The type of control message to send:

**HPR**

This is high priority. Place this message on top of the queue of messages for this terminal and control unit.

**LPR**

This is low priority. Place this message on the bottom of the queue of messages for this terminal.

**RST**

This restarts a terminal's queues.

**LNG**

This is a long (multiple segment) message. Place this message on the bottom of the queue of messages for this terminal.

**CLR**

This clears a terminal's device queue, releasing all blocks back to the TPF system.

**DLT**

This releases messages on a terminal's device queue, until an Erase/Write message is found.

**LEVEL=D0|...|DF**

This is a core block reference word (D0-DF) where the message to be sent is referenced.

### Entry Requirements

- The message to be sent must be contained in a block of storage held by the ECB on the specified level (LEVEL parameter).

- A restart or clear request does not require any data to be in the block other than a valid character count (greater than 5 and less than the block size) and the LIT (ln/ia/ta) of the terminal whose queues are to be restarted or cleared.
- For long multisegment messages (LNG option), the forward chain field of each message block must contain the address of the block containing the next segment except for the last block, which should contain zeroes.
- The message character count in the message block must include the line number (ln/ia/ta), the number of text characters in the message segment, and the end of message sequence (for example, #EOM or #EOI).
- The LIT (ln/ia/ta) must address an existing 3270 local line (symbolic line number).

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The specified core block reference word (CBRW) is initialized to indicate that the block of storage is no longer held.
- The contents of registers R14 and R15 are unknown. The contents of the remaining operational registers and the condition code are saved during processing of this macro.
- CE1SUG will be set to X'01' if an intervention required condition occurred on the specified device for a SNDLC with the HPR/LPR option.

## Programming Considerations

- The ECB reference register (R9) contains the address of the ECB (EB0EB) before this macro is used.
- A check is made by the control program (CP) to determine if the ECB is holding a block of storage at the specified level. If a block is not held, control is transferred to the operational system error routine.
- The block of storage containing the message to be sent is no longer available to the operational program.
- The status of the sending operation can be determined by the operational program by issuing a WAITC macro. This is not necessary for a restart, a clear, a long or a delete request.
- An intervening WAITC is necessary, if multiple SNDLC macros are to be issued for an HPR or an LPR request.
- The operational program may use the specified level immediately upon return from this macro.
- This macro is for use in the control program (CP) only.

## Examples

- High Priority  
`LABEL      SNDLC TYPE=HPR,LEVEL=D1`
- Low Priority  
`LABEL      SNDLC TYPE=LPR,LEVEL=D1`
- Restart  
`LABEL      SNDLC TYPE=RST,LEVEL=D1`
- Long  
`LABEL      SNDLC TYPE=LNG,LEVEL=D2`
- Clear  
`LABEL      SNDLC TYPE=CLR,LEVEL=D6`

## SNDLC

- Delete

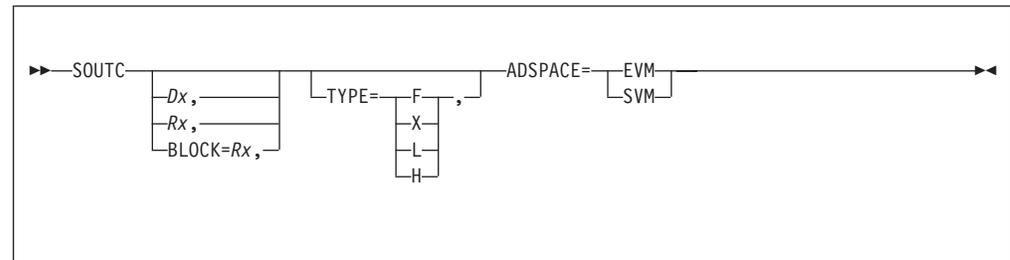
LABEL      SNDLC TYPE=DLT,LEVEL=D6

## SOUTC–Write Path Information Unit (PIU) Systems Network Architecture (SNA) Input/Output (I/O)

Use this system macro to send data to a systems network architecture (SNA) logical unit (LU) or a SNA resource.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### **Dx|Rx**

An optional data block is attached to a specified core block reference word (CBRW). The data level may be coded directly (D0|...|DF) or may be contained in a register (R0|...|R6).

This parameter is positional and, when coded, must be the first parameter. This parameter is required when the passed data block is attached to an entry control block (ECB).

#### **BLOCK=Rx**

The data to be sent resides in a data block that is not attached to an ECB. The register specified (R0|...|R6|R14) contains the address of the block. The data block must be either a 4-KB block or a system work block (SWB). The address in the specified register must be consistent with the ADSPACE parameter, ECB virtual memory or system virtual memory (SVM). This parameter is only valid when issued by C-type code.

#### **TYPE**

An optional parameter used to denote special processing.

**F** Indicates that RID to SID conversion will be bypassed.

**X** Indicates a channel contact operation (prenegotiation XID, negotiation XID, nonactivation XID contact, or discontact) is to be initiated. **No other parameters except ADSPACE parameter are valid when TYPE=X is coded.** When called from a C-type program, the ADSPACE parameter is required.

**L** Indicates that a core chained PIU is being sent.

**H** Indicates that a network layer packet (NLP) is being sent over a rapid transport protocol (RTP) connection for one of the following conditions:

- Retransmit a message that was lost in the network
- Send a message that was dequeued from the RTP output queue
- Send a high-performance routing (HPR) control message

## SOUTC

- Send an NLP, but bypass session address (SA) to resource identifier (RID) conversion
- Send an NLP that is completely built.

### ADSPACE

This parameter is required and is only valid when issued by C-type code. It is used to indicate the address space in which the code is executing in the ECB virtual memory (EVM) or system virtual memory (SVM).

### EVM

The program that processed the SOUTC is running in the EVM.

### SVM

The program that processed the SOUTC is running in the SVM.

## Entry Requirements

- When the operational program is running in the EVM, R9 must contain the ECB virtual address (EVA) of the ECB being processed.
- When the operational program is running in the SVM and the output block is attached to a data level, R9 must contain the system virtual address (SVA) of the ECB being processed.
- When TYPE=X is coded, R14 must point to a parameter list with the following format.
  - Byte 0 SDA of adjacent link station (ALS) or NCP with which channel contact operation is to be performed
  - Byte 2 Indicator byte (corresponds to SNACCIND in CW0CC) and indicates which operation is to be performed
  - Byte 3 Indicator byte for FID4 PIU or X'00' for FID2 PIU
  - Byte 4 SNA channel command word (CCW) area address for ALS (not needed for prenegotiation XID)
- When TYPE=L is coded, the following is assumed by the SOUTC processing routine:
  - The message block on the level specified, referred to as the prime block, must be in (FID1) PIU format. This means field TH1DCF is the count of the Request Header plus all the text (including those in the chained blocks). In addition, field CS1CNT (2 bytes) contains a count which is the number of bytes of text (starting at RU1BGN, beginning of the request unit) in this block. Field CS1FCH (4 bytes) of this block contains either zero, indicating this is the last block, or nonzero. When nonzero, this field contains the address of the next storage block.
  - For storage blocks pointed to by field CS1FCH of a previous block, field CS1CNT contains the text count in this block starting at field AM0TXT (AM0SG). Field CS1FCH has the same meaning as in the prime block.
  - Although no restriction is placed on the size of blocks used, all the blocks must be of the same size.
  - A maximum of 8 chained blocks may be used.
- When TYPE=F is coded, the PIU pad area must contain the session ID (PIU1SID) and CCW area index (PIU1CCWN) to be used to route this PIU.
- The format of the message passed to this macro is described in the PIUEQ DSECT.
- When TYPE=H is coded, the following lists the various conditions with their entry requirements:

- When retransmitting a message that was lost in the network:
  - The following information must be specified in the NLP pad area:
    - The NLP1SOUTC\_H field must contain X'01'.
    - The NLP1RTP field must contain the rapid transport protocol control block (RTPCB) index.
  - The block passed to the SOUTC macro must be in HPR SOUTC type-B format.
  - The header in the RTPCB table must be locked.
  - You must code ADSPACE=SVM.
- When sending a message that was dequeued from the RTP output queue:
  - The following information must be specified in the NLP pad area:
    - The NLP1SOUTC\_H field must contain X'02'.
    - The NLP1RTP field must contain the RTPCB index.
  - The block passed to the SOUTC macro must be in HPR SOUTC type-B format.
  - The header in the RTPCB table must be locked.
  - You must code ADSPACE=SVM.
- When sending an HPR control message:
  - The following information must be specified in the NLP pad area:
    - The NLP1SOUTC\_H field must contain X'03'.
    - The NLP1RTP field must contain the RTPCB index.
  - The block passed to the SOUTC macro must be in HPR SOUTC type-B format.
  - The header in the RTPCB table must be locked.
  - You must code ADSPACE=SVM.
- When sending an NLP and bypassing SA to RID conversion:
  - The following information must be specified in the NLP pad area:
    - The NLP1SOUTC\_H field must contain X'04'.
    - The NLP1RTP field must contain the RTPCB index.
    - The NLP1SA field must contain the session address.
  - The block passed to the SOUTC macro must be in HPR SOUTC type-A format.
- When sending an NLP that is completely built:
  - The following information must be specified in the NLP pad area:
    - The NLP1SOUTC\_H field must contain X'05'.
  - The following information must be specified in the PIU pad area:
    - The CCW area index must be specified in the PIU1CCWN field.
  - The block passed to the SOUTC macro must be in HPR SOUTC type-C format.

See the *TPF ACF/SNA Data Communications Reference* for more information about HPR support.

- Except when TYPE=H is coded, there is no restriction on the size of the block used for the SOUTC macro when the block is attached to an ECB. When the block is specified using the BLOCK parameter, it must be either a 4-KB block or an SWB and the logical block type must be stored in the block (in the CS1BLK field).

## SOUTC

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- For E-type code, the contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- For C-type code, the contents of all registers are preserved across this macro call.
- The specified core block reference word is initialized to indicate that a storage block is no longer held.
- The register specified in the BLOCK parameter is set to zero. The passed block will be used and released by the SOUTC service routine.

### Programming Considerations

This macro can be run on any I-stream.

### Examples

- A data block is to be attached to data level 1.  
SOUTC D1
- A channel contact operation is being initiated by a C-type program. The address space is the ECB virtual memory.  
SOUTC TYPE=X,ADSPACE=EVM
- The block attached to data level 2 is the prime block of a chain in PIU format.  
SOUTC D2,TYPE=L
- The PIU is in a block specified by register 3. The address space is the system virtual memory (SVM).  
SOUTC BLOCK=R3,ADSPACE=SVM

## SPNDC—Suspend Normal CIO Processing

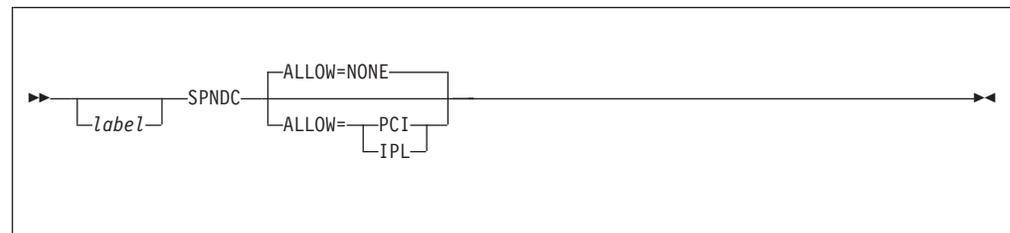
Use this system macro to suspend normal common input/output (CIO) processing and save any outstanding interrupts until CIO processing resumes. With the suspension of normal CIO, preemptive I/O (PIO) is activated.

While in the suspended state, it is possible to allow normal CIO processing for any pure PCI-I/O interrupts. You can do this by specifying the ALLOW parameter on the SPNDC macro. When ALLOW=PCI is specified, any queued or received PCI-only I/O interrupts are dispatched to their corresponding CIO device interrupt handler. Any device handler that expects PCI-only interrupts must be able to process while a system error dump is issued. This means that the device handler on the PCI-only path cannot use system services or try to lock any system resources because those services or resources may already be locked. The device handler should not issue any SERRC macros while processing a PCI interrupt.

Whenever ALLOW=NONE is changed to ALLOW=PCI, PIO dispatches any queued PCI-only interrupts before returning to the caller.

During IPL processing an ALLOW-IPL is supported by PIO to allow IPLB to communicate with the consoles on logical channel 0. Whenever ALLOW-IPL is specified, PIO dispatches any interrupts pending for devices on logical channel 0. Any symbolic device address (SDA) in the range 001 through 0FF is on logical channel 0. After completion of IPL processing, ALLOW=IPL is forced by PIO to ALLOW=NONE.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ALLOW**

Specify one of the following:

##### **NONE**

All I/O interrupts received from CIO initiated operations will be queued for dispatch until normal CIO interrupt processing is resumed. When the ALLOW parameter is not coded, this is the default value.

##### **PCI**

Any PCI-only interrupts will be immediately dispatched to the CIO device interrupt handler. Interrupts from all other devices will be handled as described under ALLOW=NONE.

##### **IPL**

This parameter is only supported during IPL. If used after IPL, PIO will assume ALLOW=NONE. ALLOW=IPL is used by IPLB to enable and dispatch any interrupts from devices on logical channel 0.

## SPNDC

### Entry Requirements

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW Key 0.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- A return code is specified in R15 as follows.

<b>Return Code</b>	<b>Meaning</b>
--------------------	----------------

0	The TPF system was not suspended previously.
4	The TPF system was suspended previously with ALLOW=NONE coded.
8	The TPF system was suspended previously with ALLOW=PCI coded.
12	The TPF system was suspended previously with ALLOW=IPL coded.

- The contents of R0, R1, R14, and R15 are unpredictable. All other registers are preserved.

### Programming Considerations

- This macro can be run on the main I-stream only.
- This macro is for use in the control program (CP) only.

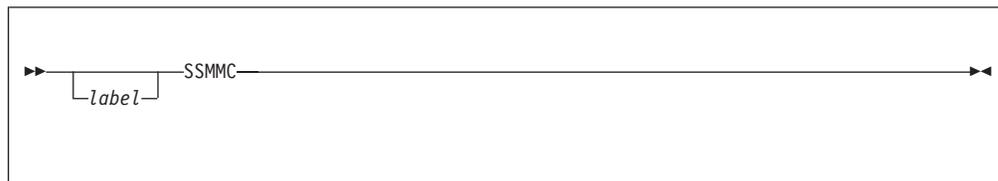
### Examples

None.

## SSMMC—Set System Mask

Use this system macro to allow an operational program to change the operating state of the central processing unit (CPU) from problem state to supervisor state. Supervisor state allows processing of privileged instructions such as set system mask (SSM) and start input/output (SIO).

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- Only use this macro with utility-type programs such as disk copy and tape reel switch that have special requirements for supervisor state.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown.
- The contents of all other registers are preserved across this macro call.
- The CPU is in supervisor state (PSW bit 15 = 0).
- The storage protect key is zero (PSW bits 8 through 11).

### Programming Considerations

- This macro can be run on any I-stream.
- Use caution when operating in this state. Any location in storage can be modified by the operational program. A program should operate in this state for the shortest possible time and then issue a LMONC macro to return to problem state.
- Supervisor state is maintained across all other macros, which can be issued by the program.

### Examples

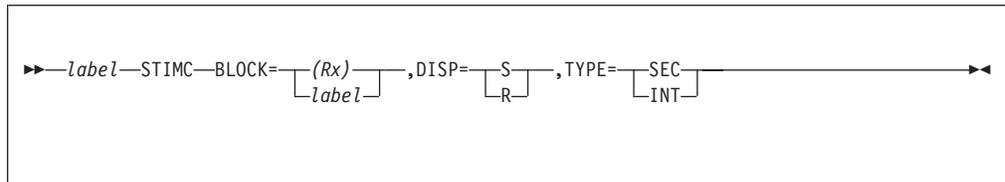
None.

## STIMC–Time-Initiated CP Routine Execution

Use this system macro to request processing of a specified routine in the control program (CP) after a specific time interval has elapsed.

The address of a core block that contains the CP routine address and time interval is placed in a table maintained by the CP. When the specific time has elapsed, the core block is placed on the ready list of the I-stream that issued the request. The request can be a one-time request or can be dispatched repetitively each time the interval has expired.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK**

Specifies the address of a core block of any valid size.

#### *(Rx)*

A register containing the address of a core block. It must be coded in parentheses.

#### *label*

The symbolic address of the location containing the core block address.

#### **DISP**

Specifies single or repetitive dispatch.

**S** The request will be dispatched once.

**R** The request will be dispatched repetitively, each time the interval has elapsed.

#### **TYPE**

Specifies the format of the time interval.

#### **SEC**

The time interval is specified in seconds.

#### **INT**

The time interval is specified in CPU timer intervals.

### Entry Requirements

Bytes 0-3 and 8-11 of the core block specified in the block parameter must be initialized with the time interval and CP routine address to be dispatched.

Bytes 0-3 The number of seconds or CPU timer intervals that will elapse before the request is dispatched.

Bytes 4-7 Used by the STIMC service routine to save the caller's I-stream address.

Bytes 8-11 The address of the CP routine to receive control when the specified time has elapsed.

Bytes 12-15 Reserved for IBM use.

The remainder of the core block is available to the user to pass data, parameters, and so on.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R0 through R15 are preserved across this macro call.
- The request is added to a table maintained by the STIMC service routine. When the specified time has elapsed, the core block will be placed on the ready list of the I-stream that issued the request. If DISP=S was coded, the request will be deleted from the table. If DISP=R was coded, the request will remain on the table for repetitive processing.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro can be run only after CTIN has initialized the timer tables and started the CPU timer.
- This macro is for use in the control program (CP) only.

## Examples

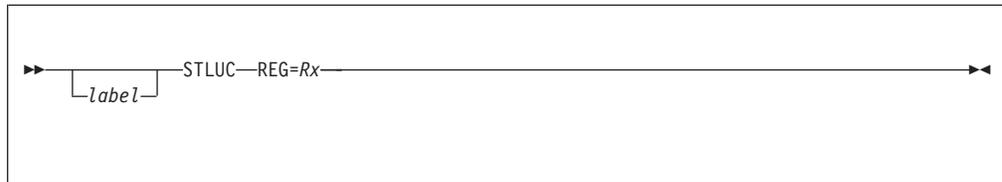
None.

## STLUC—Send LU 6.2 Message from OMT

Use this system macro to interface between the output message transmitter (OMT) and ROUTC for TPF/APPC.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

**REG=Rx**

The argument required for this parameter is a symbolic register (R0 through R7 inclusive) that points to a parameter list to be used by the TPF/APPC code.

### Entry Requirements

- This macro is for the exclusive use of TPF/APPC.
- R9 must contain the address of the ECB being processed.
- A message block must be held by the ECB at data level 0.
- The register specified by the REG parameter must point to the parameter list. The format of the parameter list follows.

Byte 0	Bit 0-7	Reserved
Byte 1	Bit 0-7	Reserved
Byte 2		Indicator 1
	Bit 0	On = First or Middle RU Off = Last RU
	Bit 1-7	Reserved
Byte 3-7		Reserved

**Note:** All reserved fields must contain zeroes.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The core block reference word (data level 0) is initialized to indicate that a storage block is no longer held.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro is restricted to system use for TPF/APPC.

## Examples

None.

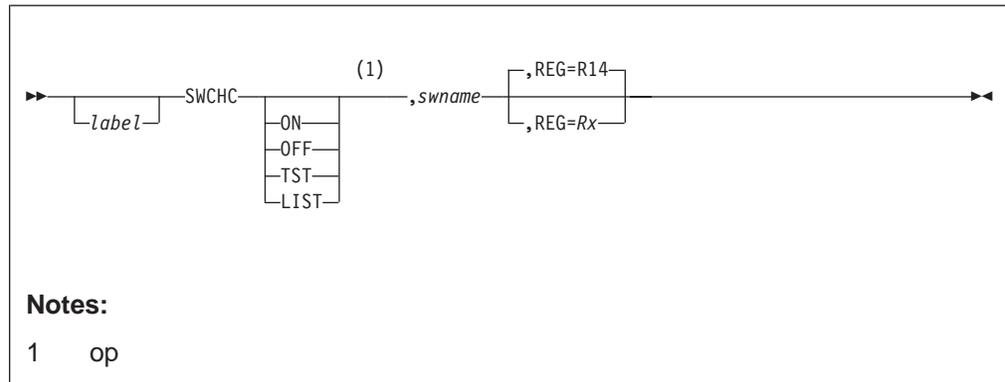
## SWCHC–Set and Test Lethal Utility Switch

Use this system macro to set, reset, and test the bit switches contained in the CK9@SW system control field in keypoint B. These bit switches are interrogated by the state change modules to determine if a state change request can be honored. The bit switches serve the same function formerly provided by the global area set of 32 switches within the area labeled @SWITCH. The switches in @SWITCH are still available but can only be used by user-defined application programs.

All bit positions formerly associated with the system utilities such as capture and restore are now contained in CK9@SW. All bit positions formerly associated with user-defined application programs have been retained in @SWITCH. Bits for all other utilities in @SWITCH are now available for use by user-defined application programs.

All of the switches in CK9@SW and in @SWITCH are checked at cycle change request time.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*op* The operation to be performed on the switch. It must be one of the following:

**ON**  
Turn the switch on

**OFF**  
Turn the switch off

**TST**  
Test the switch

**omitted**  
Test the switch (without issuing a CINFC call for keypoint B addressability, which is valid only when a previous SWCHC with the TST option has been processed).

**LIST**  
Produce only a list of EQU statements defining each bit within CK9@SW.

*swname*

This is the switch name of the bit switch contained within CK9@SW. Following is the list of valid names:

<b>SW@RECP</b>	Recoup Utility
<b>SW@SFT</b>	Selective File Trace Utility
<b>SW@FIL</b>	Record Cache Subsystem (RCS) File
<b>SW@ENA</b>	Record Cache Subsystem (RCS) Enable
<b>SW@IMP</b>	Record Cache Subsystem (RCS) Implement
<b>SW@DCOL</b>	Data Collection Utility
<b>SW@RTT</b>	Real Time Trace Utility
<b>SW@CAPT</b>	Capture Utility
<b>SW@REST</b>	Restore Utility
<b>SW@SCPY</b>	SON Copy
<b>SW@CCE</b>	3705 Dump
<b>SW@DBR</b>	Database Reorganization Utility
<b>SW@CCP</b>	CCP Trace
<b>SW@LKTR</b>	Link Trace

The 32 remaining bit switches are named for the bit they represent, such as SW@02 or SW@03. If a new utility is to be monitored during cycle change in addition to those currently being monitored, the associated switch should be renamed to reflect the new function symbolically.

**REG=R14|Rx**

This is the designated register to be loaded with the address of the selected switch bank. If omitted, the register defaults to R14. If present, the register must have been loaded with the appropriate SS/SSU ID.

## Entry Requirements

- The bits in CK9@SW can only be used by the system control program (CP) of the main I-stream.
- R9 must contain the address of the ECB being processed.
- When the request is ON or OFF, part of the generated code includes a CINFC macro call with the W (write) option. A TST request generates a CINFC macro call with an R (read) option. If the fast-path form of the macro is desired, a register preloaded with the appropriate SS/SSU ID must be specified as the REG parameter.

## SWCHC

- If several switches are to be tested, the TST parameter may be omitted from all but the first macro call.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of registers R0 through R13 are preserved across this macro call.
- The designated register or the default register, R14, points to the appropriate switch bank.
- The designated bit-switch has been turned ON or OFF, or in the case of TST, the condition code is set according to the result of a test-under-mask instruction.

## Programming Considerations

This macro can be run on the main I-stream only. This macro is intended for use on behalf of system utility programs (as distinct from utilities on behalf of user applications).

## Examples

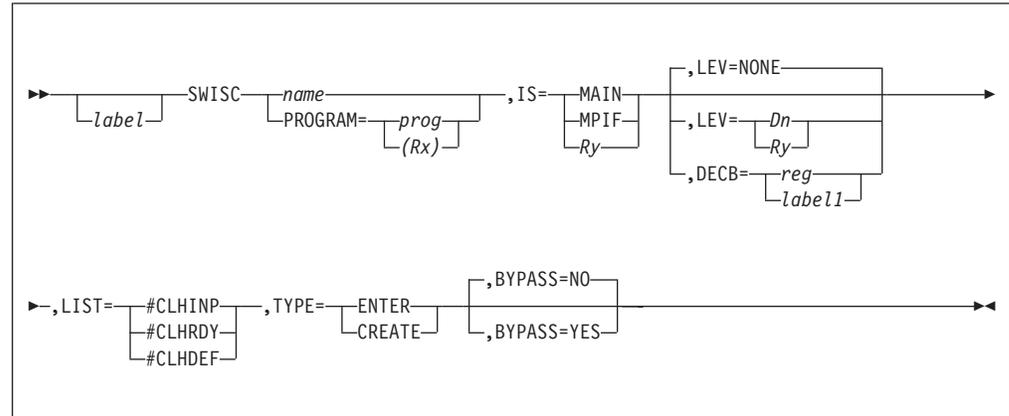
None.

## SWISC—Switch Entry to Another I-Stream

Use this system macro to to:

- Switch an existing entry control block (ECB) to another I-stream
- Create a new ECB on another I-stream.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*name*

The name of the program to activate (same as for ENTDC macros).

#### **PROGRAM**

The name of the program to activate.

*prog*

The name of the program to be activated.

**(Rx)**

A register (R0 through R7) that contains the address of where the program name can be found.

#### **BYPASS**

Specify one of the following:

##### **NO**

The default. The IS value is limited to the I-streams usable by applications, as controlled by the ZCNIS command.

##### **YES**

The IS value is limited to all the I-streams online (active) in the CPC.

#### **DECB=(reg)|label1**

A register (R0–R7) or a label containing the address of a data event control block (DECB), which contains the data to be passed to the new ECB.

When a DECB is specified, the data block is attached to data level D0 of the created ECB.

**IS** Specify one of the following:

##### **MAIN**

The main I-stream is the target.

## SWISC

### **MPIF**

The MPIF I-stream is the target.

*Ry* The specified register, R0 through R7, is set to the target I-stream number. At run time the register must contain the I-stream number where the ECB is to be created or switched. If the register contains zero, load balancing is used and the ECB is created or switched to the least busy I-stream.

### **LEV**

Specify one of the following:

#### **NONE**

No data level is to be passed to the new ECB.

#### *Dn*

The data on data level *Dn* is to be passed to the new ECB.

*Ry* The data level equate value specified in *Ry* is to be passed to the new ECB.

When a data level is specified by either *Dn* or *Ry*, the data block is attached to data level D0 of the created ECB.

### **LIST**

Specify one of the following:

#### **#CLHINP**

Will get the entry added to the input list.

#### **#CLHRDY**

Will get the entry added to the ready list.

#### **#CLHDEF**

Will get the entry added to the deferred list.

### **TYPE**

Specify one of the following:

#### **ENTER**

This entry is switched through an ENTDC macro.

#### **CREATE**

An entry is created on the dispatch list specified by the LIST parameter.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- When TYPE=CREATE is specified R14 must have the length of passed data and R15 must have the address of passed data subject to the restrictions of Create macros. A variable number of bytes (1–104) is passed to the created ECB work area.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be run on any I-stream.
- Use of the SWISC macro is not supported in the control program (CP).

- If TYPE=ENTER, this entry will be switched to the target I-stream and all program nesting will be cleared when control is passed to the new program.
- The value of LIST is restricted to those dispatch list values defined in CLHEQ.
- If LEV or DECB is coded for TYPE=CREATE, the data level or DECB specified must have an attached block. This block will be unhooked from the calling ECB on return from the macro service.
- No I/O should be outstanding when calling this macro.
- The system functions required to act on physical I-streams must specify BYPASS=YES.
- If the target I-stream is zero, the TPF scheduler will be invoked.
- In addition to the normal macro trace information the SWISC trace entry will contain the activation type, ENTER or CREATE, the name of the target program and the target I-stream number.
- If the data event control block address specified by the DECB parameter does not refer to a valid DECB, control is transferred to the system error routine and the entry is exited.
- If you use this macro to create an ECB that will enter a dynamic load module (DLM) with an entry point defined by the C language main function, the TPF system assumes that any core block attached to data level 0 (D0) contains a command string that will be parsed into argc and argv parameters for the main function. See *TPF Application Programming* for more information about the main function.

## Examples

This example creates a new ECB that initially enters the WXYZ segment.

```
*****
* SWISC to WXYZ
*
* Registers:
*
* 1. R5 is set to point to the name 'WXYZ'.
*
* 2. R3 is set to zero to request that load balancing will schedule
*    the created ECB to run on the least busy I-stream.
*
* 3. A core block is set up on data level D5. A copy of this core
*    block will be attached to the created ECB on D0. R4 is set up
*    to point to the core block's CBRW.
*
* 4. R14 and R15 are set up to contain the length and address of the
*    the parameter string. The created ECB will be initialized with
*    this string starting in EBW000.
*
* 5. The SWISC macro specifies that the new ECB will be added to the
*    CPU Ready list for the selected I-stream.
*****
      LA      R5,NEWPGM          Use PROGRAM=R5 interface to specify
*                                     the first program segment that the
*                                     created ECB will enter.
*
      SLR     R3,R3              Use IS=R3 interface to select load
*                                     balancing.
*
      GETCC  D5,L0              Get core block to pass via SWISC.
      MVC    0(MSGSIZ,R14),MSG  Move message into core block.
      LA     R4,CE1CR5          Use LEV=R4 interface to pass block.
*
      LA     R14,PARMSIZ        Length of parameter.
      LA     R15,PARM           Address of parameter.
```

# SWISC

```
*
      SWISC TYPE=CREATE,
          PROGRAM=(R5),
          IS=R3,
          LEV=R4,
          LIST=#CLHRDY,
          BYPASS=NO
*****
NEWPGM  DC   C'WXYZ'
MSG      DC   C'HELLO TO WXYZ FROM ABCD'
MSGSIZ  EQU  *-MSG
PARM     DC   C'THIS ECB WAS CREATED BY SWISC'
PARMSIZ EQU  *-PARM
```



## SYCON

- SYCON should be used to contain all configuration-dependent constants and equates.
- SYCON generates no branches. If SYCON is called to generate a table inline, it is the user's responsibility to branch around the table.
- When the System Initialization Package (SIP) is used, the key SYCON entries are initialized automatically. See *TPF System Generation* for more information.
- Any segment that references SYCON becomes configuration-dependent. To avoid unnecessary assemblies, SYCON should not be called when the needed information can be provided by the CINFC or CONKC macros.
- In order to minimize the number of online programs that must be reassembled should SYCON values change, the CONKC macro, the configuration constants data record (CONKC), CINFC macro and the communications keypoint (CK6KE) have been provided to permit real-time access to SYCON-specified values.

## Examples

Macro Call (define equates only)

```
SYCON
```

Macro Call (define equates and DASD Configuration Table)

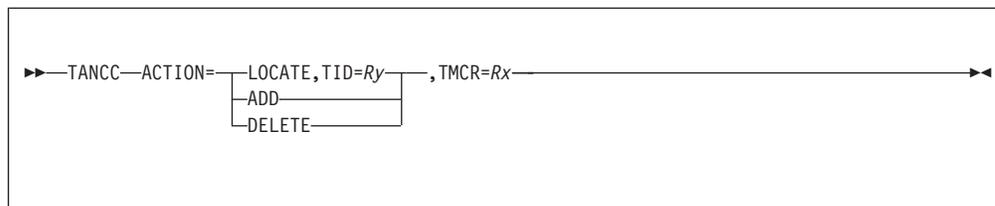
```
SYCON CON=3
```

## TANCC–Transaction Anchor Table Control (TANC)

Use this system macro to add, delete, or locate a transaction manager control record (TMCR) in the transaction anchor table (TANC).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### ACTION

Defines the type of action requested. This parameter is required. The following actions are valid:

##### ADD

To add a TMCR pointer to the TANC.

##### DELETE

To remove a TMCR pointer from the TANC.

##### LOCATE

To locate a TMCR pointer for a given transaction ID (TID).

#### TMCR=R<sub>x</sub>

A register (R0–R7, R15) that contains the address of a TMCR. This is a required input parameter when ADD or DELETE are specified, and is also a required output parameter when LOCATE is specified.

#### TID=R<sub>y</sub>

A register (R0–R7, R15) that contains the address of the TID that is used to locate the TMCR pointer in the TANC. This parameter is required only when locating a TMCR pointer.

### Entry Requirements

- When adding or deleting a TMCR pointer, the register must point to the TMCR that will be added to or removed from the TANC.
- When locating a TMCR pointer, the register must point to the TID that will be used to locate the TMCR in the TANC. R9 must point to a valid entry control block (ECB).

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R0–R7 are preserved across this macro call.
- The condition code (CC) is not preserved across the macro call.
- The macro service routine will set R14 to contain a hexadecimal return code that can be used by the calling program.

## TANCC

R14=00	Request is completed successfully.
R14=-2	Input TMCR parameter is 0.
R14=-3	Input TID parameter is 0.
R14=-4	Adding a TMCR pointer was not successful because the TID in the input TMCR already exists.
R14=-5	Deleting or locating a TMCR pointer was not successful because no matching TID exists in the TANC.

## Programming Considerations

- This macro can be run from any I-stream.
- This macro can be run by both ECB-controlled programs and control program (CP) programs.

If a log takeover occurs, there will be more than one TANC in the TPF system. The TANCC macro will use a processor index value that is kept in the TMCR (for adding or deleting) or in the ECB (for locating) to perform the function on the correct TANC.

## Examples

The following example shows how to add a TMCR into the TANC.

```
TANCC ACTION=ADD, TMCR=R2
```

## TASBC—Turn Off Time Available Supervisor Switch

Use this system macro to turn off the time available supervisor (TAS) demand switch in the system task dispatcher. Turning off the TAS demand switch indicates that there are no low priority tasks to be handled by the time available supervisor.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format

<pre> ▶▶—<i>label</i>—TASBC—————▶▶ </pre>
---

*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- The TAS demand switch remains reset until set by the TASTC macro.

### Examples

None.

## TASTC—Turn on Time Available Supervisor Switch

Use this system macro to turn on the time available supervisor (TAS) demand switch in the system task dispatcher. Turning on the TAS demand switch activates the supervisor that schedules the processing of low priority tasks such as teletype reservation messages.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format

<pre> ▶▶—<i>label</i>—TASTC—◀◀ </pre>
---------------------------------------

*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- When the TAS demand switch is set, the time available supervisor can gain control every 250 milliseconds.
- The TAS demand switch remains set until reset by the TASBC macro.

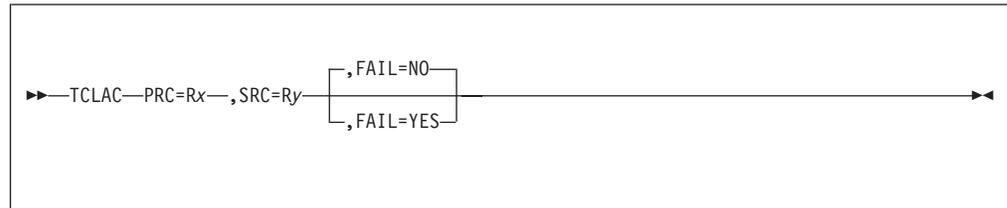
### Examples

None.

## TCLAC—Write a CLAW Error Log

Use this system macro inline to write a specified error log record using the SNAPC conventions. Common Link Access to Workstation (CLAW) control program (CP) segments use this macro.

### Format



#### PRC=R<sub>x</sub>

This required register contains the primary return code indicating the primary reason for the log error subrequest.

#### SRC=R<sub>y</sub>

This required register contains the secondary return code indicating the secondary reason for the log error subrequest.

#### FAIL

This optional parameter determines if the adapter is to be closed.

#### YES

The error log is written and the adapter is closed.

#### NO

The error log is written and the adapter is not closed.

### Entry Requirements

This macro is for use in the control program (CP) only.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of all registers, except R14 and R15, are preserved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- The FAIL option sets the fail option bit. The fail bit must be set for CLAW to fail.

### Examples

This call indicates a primary return code is in register R6 and a secondary return code is in R0. The FAIL option indicates the trace record will be written and the adapter will be closed.

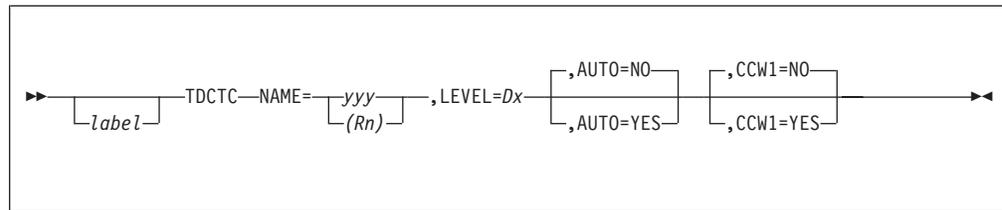
```
TCLAC PRC=R6,SRC=R0,FAIL=YES
```

## TDCTC—General Tape Data Chain Transfer

Use this system macro to address a general tape and read or write a single record into or from a specific storage area using a series of data chained channel command words (CCWs).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name may be assigned to the macro statement.

**NAME**

Specifies the symbolic general tape name. It can be:

*yyy*

A 3-character string representing a symbolic general tape name. The first two characters must be alphabetic, and the third character must be alphabetic or numeric. The first two characters cannot be RT.

**(Rn)**

The number of a register containing a pointer to the symbolic real-time tape name. *n* must be a decimal number from 0 through 7, 14, or 15.

**LEVEL=Dx**

A symbolic data level (D0–DF) must be specified.

**AUTO=NO|YES**

Specifies whether an automatic tape switch occurs for end-of-tape, end-of-volume, or hardware error conditions.

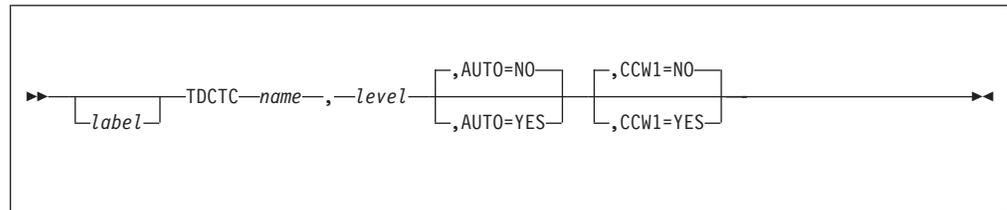
**Notes:**

1. If the tape is blocked, TPF tape support will force AUTO=YES regardless of how this parameter is entered.
2. If the tape is blocked or AUTO=YES is specified, an end-of-tape or hardware error from a tape write operation, or an end-of-volume error from a tape read operation causes an automatic tape switch to occur. End-of-file and hardware errors from tape read operations do not cause an automatic tape switch to occur.
3. If the tape is not blocked and AUTO=NO is specified, end-of-tape, end-of-volume, and hardware errors are indicated in the ECB.

**CCW1=NO|YES**

This optional parameter is used to indicate the CCW format in the specified data level. If CCW1=YES, the CCW is treated as CCW format-1, otherwise CCW format-0 is assumed. CCW1=NO is the default.

The following macro format is still supported.



*name*

A 3-character symbolic general tape name must be specified as the first parameter.

*level*

A symbolic data level (D0–DF) must be specified as the second parameter.

## Entry Requirements

- R9 must contain the address of the entry control block (ECB) being processed.
- The general tape specified by tape *name* must be open when this macro is issued.
- The first 4 bytes of the file address reference word (FARW) of the data level specified by *level* must contain the address of the channel program. The channel program must only refer to addresses above X'1000'.
- The CCWs you provide must be data chained to ensure a single block transfer.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The status of the data transfer operation is unknown.

## Programming Considerations

- This macro can be run on any I-stream.
- Although both keyword and positional parameters may be used in the same macro call this method is not recommended.
- Tape data transfer depends on various conditions:
  - For a blocked tape, a WAITC call ensures that the data has been transferred to the blocking buffer. This is true whether the tape is mounted on a buffered device or an unbuffered device.
  - For an unblocked tape mounted on a buffered device that is operating in buffered mode, a WAITC call guarantees that the data has been written to the control unit buffer.
  - For any other unblocked tape, a WAITC guarantees that the data has been written to the tape.
- For devices operating in buffered mode, a hardware error reported to the ECB may indicate a loss of buffered data. See the TOPNC macro in *TPF General Macros* for more information about the buffered mode.
- When an unusual condition is detected that causes processing of the CCW chain to be halted before completion of the last CCW, the address portion of the resultant channel status word will be stored in the last four bytes of the FARW of the data level specified by *level*.

## TDCTC

- The storage area on the CCW chain containing the data being written or read must remain unchanged until the operation is complete.
- This macro should not be run to write heap storage areas to a blocked tape.
- The command code in the first CCW must be either a Write, Read, Read Backwards, or Synchronize. This macro cannot be used to run a Read Backwards command to a blocked tape. If this is attempted with a blocked tape, a system error is issued.
- For write operations, the Suppress Length Indication flag should be set in the last CCW.
- For write operations to a blocked tape, the maximum byte count of the record being written is 32 752. For write operations to an unblocked tape, the maximum byte count of the record being written is 65 535. If an attempt is made to write a record longer than the maximum length, a system error is issued.
- For write operations, if the tape is mounted in unblocked mode, the mode of operation (buffered or Tape Write Immediate) is determined by the entry in the tape status table. If the tape is mounted in blocked mode, the mode of operation is always buffered.

For write commands issued to a 3480 device, bit 0 of the secondary indicator byte in the corresponding tape status table entry will determine the operating mode (buffered or tape write immediate). This bit is initially set when the tape is opened using the TOPNC macro.

## Examples

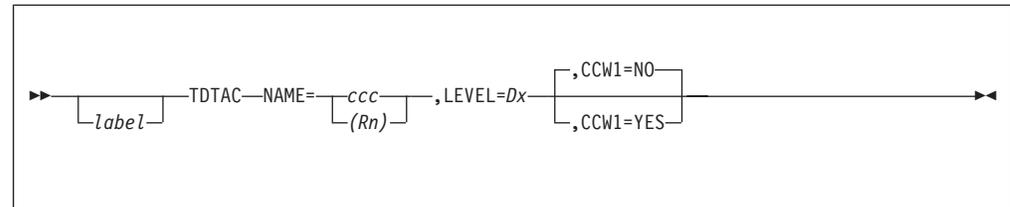
None.

## TDTAC–General Tape Data (GDS) Transfer

Use this system macro to address a general tape and perform a single data transfer operation.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic general tape name. It can be:

*ccc*

A 3-character string representing a symbolic general tape name. The first two characters must be alphabetic, and the third character must be alphabetic or numeric. The first two characters cannot be RT.

**(Rn)**

The number of a register containing a pointer to the symbolic general tape name. *n* must be a decimal number from 0 through 7, 14, or 15.

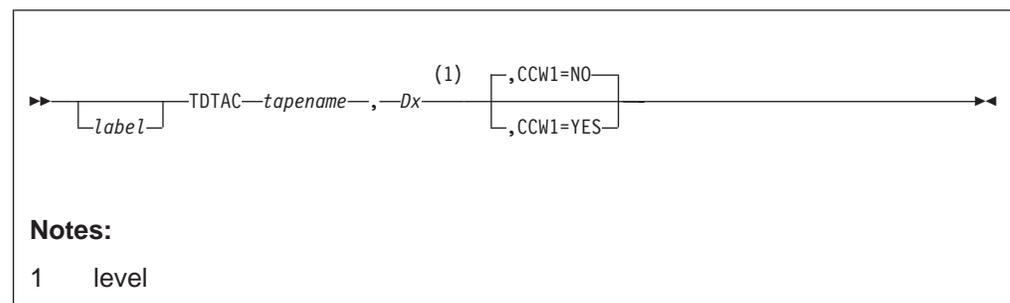
**LEVEL=Dx**

A symbolic data level (D0–DF) must be specified.

**CCW1=NO|YES**

This parameter is used to indicate the format of the channel control word (CCW) in the specified data level. If `CCW1=YES`, the CCW is treated as CCW format-1, otherwise CCW format-0 is assumed. NO is the default.

The following macro format is still supported.



*label*

A symbolic name may be assigned to the macro statement.

## TDTAC

### *tapename*

A 3-character symbolic general tape name must be specified as the first parameter.

### *level*

A symbolic data level (D0-DF) must be specified as the second parameter.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The file address reference word (FARW) for the data level specified by the LEVEL parameter (*level*) must contain the data transfer CCW. The channel program must only refer to addresses above X'1000'.
- The general tape specified by the NAME parameter (*tape\_name*) must be open when this macro is issued.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The status of the data transfer operation is unknown.
- During normal processing of this function, the CCW contained in the FARW is not modified. If an incorrect record length is found, the length field will be in the last 2 bytes of the FARW on the data level specified by the macro.
- During the processing of blocked tapes when a condition such as end-of-volume (EOV) or permanent error occurs, a tape switch will automatically occur. For unblocked tapes the automatic tape switch will not occur. For blocked tapes AUTO=YES is implied and for unblocked tapes AUTO=NO is implied. See "TDCTC—General Tape Data Chain Transfer" on page 478 for more information. Also see *TPF General Macros* for more information about TSYNC and the synchronize tape macro.
- The storage area containing the data that is being written or read must remain unchanged until the operation is complete.

## Programming Considerations

- This macro can be run on any I-stream.
- Although both keyword and positional parameters may be used in the same macro call this method is not recommended.
- For blocked tapes, a WAITC call will ensure that the data has been transferred to the blocking buffer.
- For buffered devices, a WAITC call guarantees that the data has been written to the control unit buffer.
- For non-blocked tapes on devices operating in tape write immediate mode, a WAITC guarantees that the data has been written to the tape.
- This macro executes independently of normal tape macro handling and volume switching facilities. When an unusual condition such as end-of-file, end-of-tape, or hardware error is encountered, return is made to the operational program through the system error WAITC return mechanism. All subsequent I/O requests are processed normally and are also flagged appropriately at WAITC return. It should be noted that write operations are allowed to complete when an end-of-tape condition is detected, and the operation is appropriately flagged to

indicate this condition. For this macro only, an end-of-tape condition on a write operation is flagged by the same indicator bit as an end-of-file condition on a read operation.

- For device operating in buffered mode, a hardware error reported to the ECB may indicate a loss of buffered data. See *TPF General Macros* for more information about TOPNC and the open a general tape macro, as well as information about buffered mode.
- The storage area containing the data that is being written or read must remain unchanged until the operation is complete.
- This macro should not be used to write Heap storage areas to a blocked tape.
- For write operations to a blocked tape, the maximum byte count of the record being written is 32 752.

For write operations to an unblocked tape, the maximum byte count of the record being written is 65 535. If an attempt is made to write a record longer than the maximum length, a system error is issued.

- The minimum record size that can be written or read is 16 bytes.
- For write and sense commands, the suppress length indication flag is automatically set by the control program (CP).
- This macro cannot be used to run a Read Backwards command to a blocked tape. If this is attempted with a blocked tape, a system error is issued.
- No check is made by the control program (CP) to see if the command is supported by the device. For example, a command code to read configuration data can be sent to a 3480 tape device, even though it is undefined for that device type.
- The supported channel commands for this macro follow here.

Hexadecimal Code	Channel Command Action on Tape Drive
01	Write
02	Read forward
03	No operation (NOP)
04	Sense
0C	Read backward
22	Read block ID
24	Read buffered log
34	Sense path group ID
43	Synchronize
4F	Locate block
E4	Sense ID
64	Read device characteristics
77	Perform subsystem function
9F	Load display
AF	Set path group ID
B7	Assign
C7	Unassign
E3	Control access

## TDTAC

FA

Read configuration data.

## Examples

None.

---

## TERMC-Kill a Threaded Process

Use this system macro to allow an entry control block (ECB) that is not in a thread process to get information about the number of threads in a process, and to force the ECBs running inside the process to exit. TERMC will pass an event name to the process to allow threads to post the event when exiting.

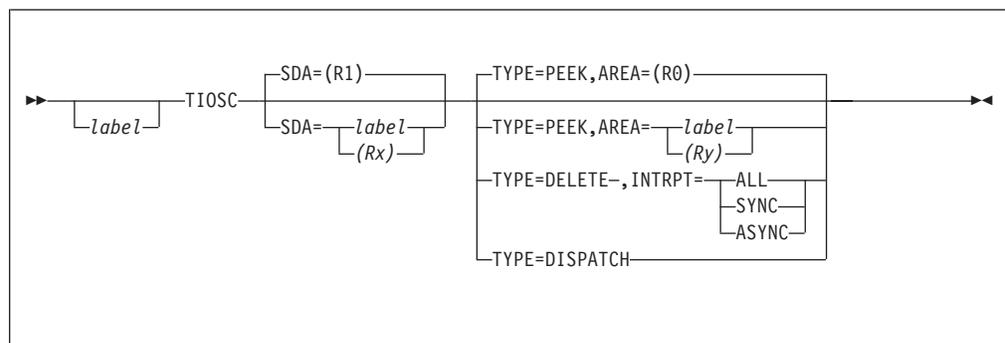
**Note:** Syntax and parameter descriptions are not provided because this is an object code only (OCO) macro.

## TIOSC—Test Input/Output (I/O) Service

Use this system macro to request the following services.

Service	Description
PEEK	It looks at the synchronous and asynchronous interrupts queued for a specific symbolic device address (SDA). The data for the queued interrupts is placed in an area provided by the requestor in accordance with the DCTTIO DSECT.
DISPATCH	It dispatches the synchronous and asynchronous interrupts queued for a specific SDA. When an interrupt is dispatched, it is sent to the common input/output (CIO) device interrupt handler as specified in the mount request.
DELETE	It deletes the synchronous and asynchronous interrupts for a specific SDA.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **SDA**

This is either the label of a halfword field that contains the SDA, or a register that contains the SDA in bytes 2 and 3 and zeros in bytes 0 and 1. The default is SDA in R1.

#### **TYPE**

This is the function that is requested by the caller:

- PEEK (default)
- DISPATCH
- DELETE.

#### **AREA**

AREA is only valid for TYPE=PEEK. AREA is either the label assigned to an area defined by the DCTTIO DSECT or a register that contains the address of the area. If AREA is not specified, the default register is R0.

#### **INTRPT**

This is the interrupt type to be deleted. INTRPT is required for TYPE=DELETE.

#### **ALL**

Both synchronous and asynchronous interrupts

#### **SYNC**

Synchronous interrupt only

**ASYNC**

Asynchronous interrupt only

**Entry Requirements**

- The TPF system must be disabled for interrupts.
- The TPF system must be running with PSW key 0.

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- R0 through R15 will be unchanged.
- A condition code will be set depending on the value specified for the TYPE parameter.
  - PEEK, which is the area pointed to by the AREA parameter and described by the DCTTIO DSECT will be filled in. The condition code will be set as follows.

<b>Condition Code</b>	<b>Meaning</b>
0	No active request
1	Interrupt information returned
2	Device currently active
3	A request that is not valid
– DISPATCH for which the condition code will be set as follows.	

<b>Condition Code</b>	<b>Meaning</b>
0	No active request or operation performed
2	Device currently active
3	A request that is not valid
– DELETE for which the condition code will be set as follows.	

<b>Condition Code</b>	<b>Meaning</b>
0	No active request or operation performed
2	Device currently active
3	A request that is not valid

**Programming Considerations**

- This macro can be run on the main I-stream only.
- If the TYPE=DISPATCH, the macro user should not be holding any locks that may be required by the specified SDA device interrupt handler.
- If only one of the interrupt types is to be dispatched, the other, if it exists, must first be deleted.
- This macro is for use in the control program (CP) only.

**Examples**

None.

## TMSLC–Time Slice an ECB

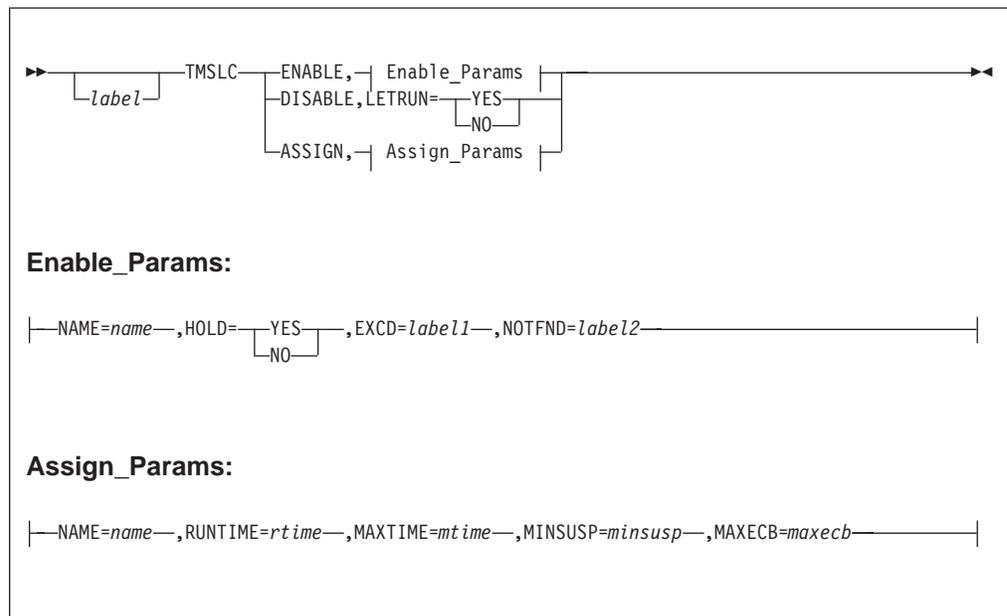
Use this system macro to to:

- Enable or disable time slicing for an entry control block (ECB)
- Assign time-slicing attributes to a time-slice name.

When an ECB is enabled for time slicing, it loses control at defined time intervals. This allows other tasks in the TPF system to receive control.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### ENABLE

Allow the ECB to time slice.

When the entry control block (ECB) reaches the RUNTIME value without giving up control, it gets suspended. The ECB will remain suspended for the value specified in the MINSUSP parameter.

**Note:** RUNTIME and MINSUSP are values defined with the NAME parameter. If you specify HOLD=NO and the ECB is holding a resource, the ECB will not get suspended.

#### NAME=name

The time-slice name parameter is a 1–8 character name. The time-slice name has four time-slice parameters associated with it. The four parameters are RUNTIME, MAXTIME, MINSUSP, and MAXECB.

The predefined time slice names and values of the parameters associated with them are given in Table 8.

**HOLD=YES|NO**

Indicates if the ECB can be suspended while holding a resource (FIWHC, CORHC, ENQC, EVNWC, TASNC, SYNCC LOCK).

**Note:** This does not apply to resources that are held with the \$LOCKC macro.

**EXCD=*label1***

The address to which control is passed if the maximum number of ECBs that can be time sliced are already active for the specified time-slice name.

**NOTFND=*label2***

The address to which control is passed if the NAME parameter contains a name that is not recognized.

**DISABLE**

Disable time slicing. Return the ECB to normal operation.

**LETRUN**

Used with the DISABLE parameter to determine if the ECB continues to run or gives up control.

**YES**

The ECB will not lose control.

**NO**

The ECB will lose control unless time slicing is enabled with HOLD=NO and the ECB is currently holding a resource.

**ASSIGN**

Assign time-slicing attributes to a time-slice name.

**NAME=*name***

The time-slice name parameter is a 1 to 8 character name. The time-slice name has four time-slice parameters associated with it. The four parameters are RUNTIME, MAXTIME, MINSUSP, and MAXECB.

The predefined time-slice names and values of the parameters associated with them are given in Table 8.

**RUNTIME=*rtime***

The amount of CPU time (from 10 through 500 ms) the ECB is allowed to use without giving up control before it is time-sliced.

**Note:** This value cannot exceed the value that is generated by the system initialization program (SIP) for the application timeout counter (set by IBM to 500 ms).

**MAXTIME=*mtime***

The maximum amount of accumulated runtime (from 0 ms through 9 999 999 ms) the ECB can use before taking a system error. A value of 0 means the ECB can run indefinitely.

**MINSUSP=*minsusp***

The minimum amount of time (from through 9999 ms) that the ECB will be suspended if it is time sliced.

**MAXECB=*maxecb***

The maximum number of ECBs (from through 9999) that can be active and enabled for time slicing with this time-slice name.

## TMSLC

### Entry Requirements

- R9 contains the address of the ECB.
- This macro can only be run if the issuing program has restricted macro usage authorization.
- The ASSIGN parameter can only be used by segment CTMS.

### Return Conditions

Control returns to the next sequential instruction (NSI).

### Programming Considerations

- This macro can run on any I-stream.
- This macro cannot be run while the TPF system is in restart.
- Time slicing should not be enabled throughout the life of the transaction because there is no protection against an ECB that allows time slicing calling a segment which does not allow time slicing.
- Time slicing should be enabled only in specific CPU-intensive code paths. Time slicing is not supported across TPF services such as the following because these services may be referencing shared memory, issuing \$LOCKC macros, or holding critical resources:
  - SVC and fast-link macro calls
  - Enters to TPF real-time segments (CYYM, CYYA, BPKD, and others)
  - TPF Database Facility (TPPDF) functions.

You must ensure that time slicing is disabled before calling such system services.

- An ECB that is enabled for time slicing must never issue a \$LOCKC macro.
- An ECB enabled for time slicing must never update a global field or other storage areas that can be updated simultaneously by other ECBs.
- Coding LETRUN=YES is advantageous only in a highly repetitive loop where time slicing is enabled and disabled frequently. Otherwise, to avoid timeout errors, specify LETRUN=NO.

The ECB will exit with a system error if the ECB is not forced to give up control and the ECB continues to run for 500 ms without giving up control.
- An ECB enabled for time slicing and running with the HOLD=YES parameter can hold a resource while suspended. This can cause hang conditions. Resources held by an ECB enabled for time slicing should not be needed elsewhere.
- An ECB enabled for time slicing can exit with a system error if the HOLD=NO parameter is specified. This can occur if the ECB is holding a resource (which prevents time slicing) and has been running without giving up control for greater than the time allowed by the application timeout value (500 ms).
- The maximum amount of accumulated runtime that an ECB enabled for time slicing is allowed before exiting with a system error is not reset each time a TMSLC call with the ENABLE parameter is issued. The amount of time set with the MAXTIME parameter must allow the ECB to complete its task.
- ECBs that are suspended because of a TMSLC call will be purged during system cycle-down to 1052 state unless they have been previously identified to survive cycle-down.
- Resource control is shipped with three predefined time-slice names. Table 8 shows the predefined time-slice names and values of the parameters associated with them.

Table 8. Time Slice Name Table

Time Slice Name	Values			
	RUNTIME	MAXTIME	MINSUSP	MAXECB
IBMLOPRI	50 ms	20000 ms	1000 ms	50
IBMHIPRI	100 ms	10000 ms	100 ms	50
IBMINDEF	50 ms	0 ms	2000 ms	20

**Notes:**

1. These are the values for the time-slice name parameters as shipped by IBM.
2. IBMLOPRI, IBMHIPRI, and IBMINDEF are reserved for use by IBM.
3. To add new time-slice names, use the ZTMSL command or the TMSLC macro with the ASSIGN parameter. The ZTMSL command can also be used to display, change, and remove time-slice names.

**Examples**

- This call to the TMSLC macro:
  - Enables the ECB for time slicing based on the parameters associated with the time-slice name of BIGSORT,
  - Returns control to label WARNUSER if too many ECBs are enabled for time slicing under the time-slice name BIGSORT,
  - Indicates the ECB is not allowed to be suspended while holding a resource, and
  - Branches to BADNAME, if the TMSLC macro does not recognize the time-slice name BIGSORT.

```
TMSLC  ENABLE,NAME=BIGSORT,HOLD=NO,EXCD=WARNUSER,NOTFND=BADNAME
```

- This call to the TMSLC macro creates an entry for the IBMXYZ name in the time-slice name table during system restart. The entry is initialized with the following attributes:
  - Set the maximum time that ECBs can run continuously to 300 ms.
  - Set the maximum amount of accumulated runtime that ECBs can use to 5000 ms.
  - If suspended because of a time slice, suspend ECBs for 1000 ms.
  - Allow no more than 500 ECBs to be enabled for time slicing with the IBMXYZ time-slice name.

```
TMSLC  ASSIGN,NAME=IBMXYZ,RUNTIME=300,MAXTIME=5000,MINSUSP=1000,MAXECB=500
```

- This call to the TMSLC macro does not allow the ECB to be time sliced. Upon return, the ECB will not be able to be time sliced and will act as a normal transaction ECB.

The ECB will lose control unless it was enabled with HOLD=NO and is currently holding a resource. Time-slicing is disabled when updating global storage, when entering \$LOCKC macro calls, when holding performance-critical records, and so on.

```
TMSLC  DISABLE,LETRUN=NO
```

## TMTKC—Get the Unique Token for the Current Transaction

Use this system macro to get the unique token for the current transaction.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **TOKEN=token**

Gets the unique token for the current transaction, where *token* is the symbolic name of a field or a register pointing to a field. The field is an 8-byte storage area in which the token will be stored. If you specify a register, the register must be enclosed in parentheses and in the range R0–R7.

### Entry Requirements

None.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The storage area pointed to by the TOKEN parameter contains one of the following:
  - A nonzero unique transaction token for the current transaction in which the entry control block (ECB) is running.
  - A zero, which indicates that the ECB is not currently in an active transaction.
- The contents of R14 and R15 cannot be predicted.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro does not verify if the ECB has the protection key to write to the storage area where the token will be stored. The calling program must ensure that the area specified with the TOKEN parameter can be updated.

### Examples

The following example gets the unique token for the current transaction and stores the token in the area to which R5 points.

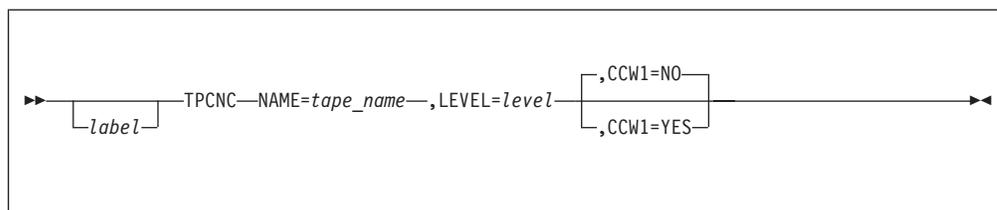
```
TMTKC TOKEN=(R5)
```

## TPCNC–Tape Control

Use this system macro to process certain tape control channel command words (CCWs) that are not included in the normal macro set (for example, forward space block).

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



*label*

A symbolic name may be assigned to the macro statement.

**NAME=tape\_name**

A 3-character symbolic general tape name must be specified as the first parameter.

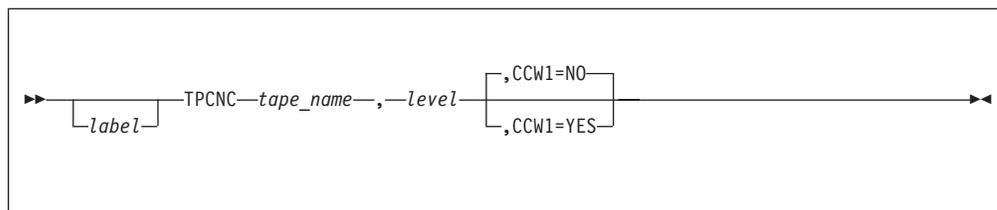
**LEVEL=level**

A symbolic data level (D0-DF) must be specified as the second parameter.

**CCW1=NO|YES**

This parameter is used to indicate the format of CCW in the specified data level. If CCW1=YES, the CCW is treated as CCW format-1, otherwise CCW format-0 is assumed. CCW1=NO is the default.

The following format is still supported.



*label*

A symbolic name may be assigned to the macro statement.

*tape\_name*

A 3-character symbolic general tape name must be specified as the first parameter.

*level*

A symbolic data level (D0-DF) must be specified as the second parameter.

**CCW1=NO|YES**

This parameter is used to indicate format of CCW in the specified data level. If CCW1=YES, the CCW is treated as CCW format-1, otherwise CCW format-0 is assumed. The default is CCW1=NO is the default.

## TPCNC

### Entry Requirements

- R9 must contain the address of the entry control block (ECB) being processed.
- The general tape specified by *tape\_name* must be open when this macro is issued.
- The file address reference word (FARW) for the data level specified by *level* must contain a modified tape control CCW. The format of the CCW follows.

CE1FAX	Byte 0	Command Code
	Bytes 1-3	Not Used
CE1FMx	Byte 4	Command Flags
CE1FCx	Byte 5	Not Used
CE1FCx	Bytes 6 and 7	Count of Operation Executions

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The status of the control operation is unknown.

### Programming Considerations

- This macro can be run on any I-stream.
- To ensure completion of the control operation, a WAITC macro should be coded after this macro.
- Only the data chain, suppress incorrect length indication, and suspend flags may be set in the CCW.
- During normal processing of this macro the CCW contained in the FARW is not modified. If an unusual condition is encountered, however, the operation count in the CCW will be adjusted to reflect the actual number of operations successfully processed.
- When an unusual condition such as end-of-file or end-of-tape is encountered return is made to the operational program through the system error WAITC return mechanism. All subsequent I/O requests are processed normally and are also flagged appropriately at WAITC return.

### Examples

None.

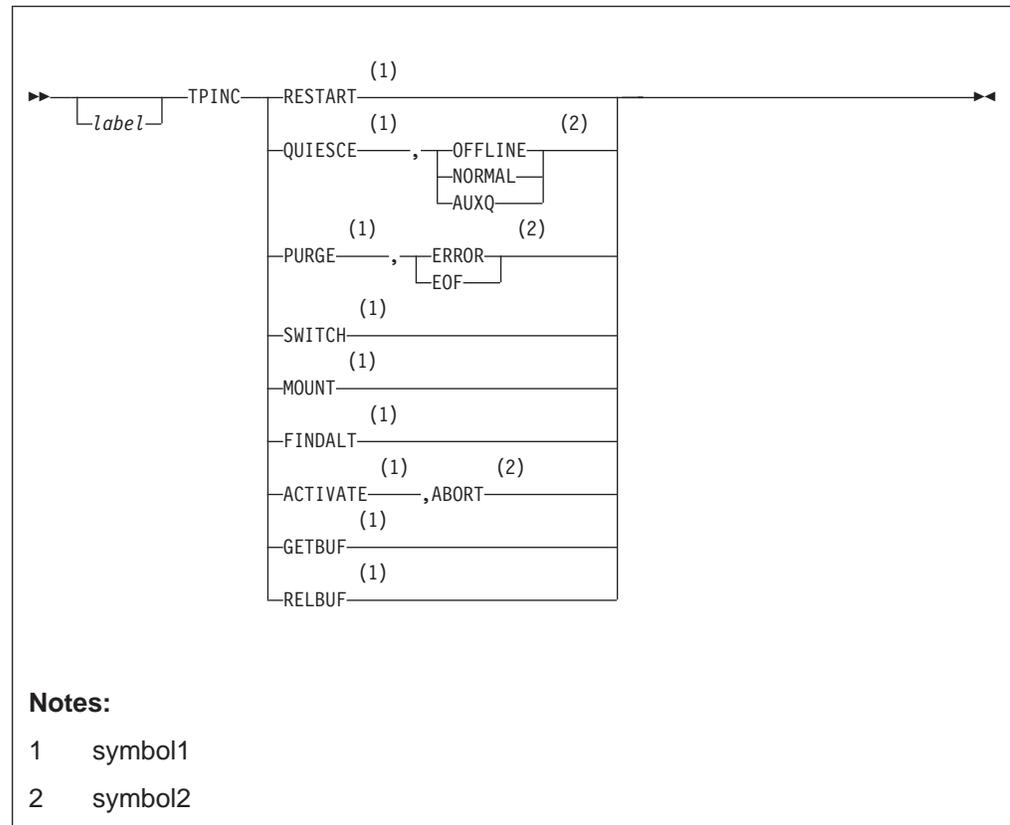
## TPINC–Special Tape Interface

Use this system macro to provide an interface to certain tape control program (CP) routines for entry control block (ECB)-controlled segments in the tape handling area. These routines allow the ECB routine to:

- Restart input/output (I/O) operations for a specific tape
- Quiesce I/O operations for a specific tape
- Purge I/O requests for a specific tape to a specific postinterrupt routine
- Initiate a CP-controlled tape switch from active to standby for a specific tape
- Initiate processing to mount an alternate (ALT) tape on the specific device
- Locate an ALT tape when required for a tape switch
- Get blocking buffer for a blocked tape
- Release blocking buffer for a blocked tape.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



*label*

A symbolic name can be assigned to the macro statement.

*symbol1*

An interface function must be specified as symbol1. The interface functions are:

**RESTART**

Restart I/O operations for the specified tape.

**QUIESCE**

Set appropriate tape status as specified by *symbol2* and await completion of outstanding I/O operations for the specified tape.

**PURGE**

Purge all outstanding I/O requests for the specified tape to the postinterrupt routine specified by *symbol2*.

**SWITCH**

Initiate a tape switch from active to standby for a specified tape.

**MOUNT**

Initiate processing to mount an ALT tape on the specified device.

**FINDALT**

Locate an ALT tape for a tape switch when no standby is available.

**ACTIVATE**

Activate a specific ECB or all ECBs currently on the tape wait list.

**GETBUF**

Get the blocks required for the buffer of a blocked tape.

**RELBUF**

Release the blocks from the buffer of a blocked tape.

*symbol2*

A tape status must be specified as the second parameter when the QUIESCE function is specified as *symbol1*. A postinterrupt routine must be specified as the second parameter when the PURGE function is specified as *symbol1*.

The tape statuses for the QUIESCE function are:

**OFFLINE**

Mark the specified tape offline prior to quiescing the tape.

**AUXQ**

Mark the specified tape auxiliary queued prior to quiescing the tape. *Auxiliary queued* is a term that describes temporary suspension of normal tape I/O operations due to conditions unique to tape processing. For example, when an end-of-volume condition is detected for an output tape, special handling is required to write trailer records, rewind the tape, and initialize a new tape.

When a tape is auxiliary queued, normal I/O requests are queued but no I/O is performed. Tape handling routines can force processing, however, by requesting a bypass of this suspension. An I/O request of this type is said to be a *bypass auxiliary queue request*.

**NORMAL**

Do not alter the status of the specified tape prior to quiescing the tape.

The postinterrupt routines for the PURGE function are:

**ERROR**

Hardware error post-interrupt routine.

**EOF**

End-of-file post-interrupt routine.

The option available with the ACTIVATE function is:

**ABORT**

The specified ECB will be removed from the tape wait list and exited.

**Entry Requirements**

- R9 must contain the address of the ECB being processed.
- CE1TAP must contain the module number of the tape specified by symbol1.
- The entry must have seized the tape specified by symbol1 (that is, CPMTIF must have been set to X'FF').

**Return Conditions**

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The status of the operation is unknown.
- If TPINC QUIESCE,NORMAL or TPINC QUIESCE,OFFLINE was coded, then CE1TIN will have been set to X'FF'. Otherwise, CE1TIN will have been set to X'00'.

**Programming Considerations**

- This macro can be run only on the main I-stream.
- To ensure completion of this operation, a WAITC macro must be issued. Usage of the WAITC error branch depends on the function specified by symbol1 as described below.
- A check is made by the tape control program (CP) to ensure that the segment issuing the macro is an ECB-controlled system segment (that is, the first character of the segment name is C). If this condition is not met, control is transferred to the system error routines.
- The following considerations are specific to the function specified by symbol1.

***Purge***

Any I/O request for the specified device that is already in progress at the time the macro is issued will be allowed to complete. Any I/O requests for the specified device that are not already in progress will be ended without I/O initiation. After completion of any I/O operation already in progress, the ECB I/O counter will be decremented. No error indicators will be set.

The module queue is considered to have been successfully purged if no I/O requests have been added to the queue between the time the TPINC PURGE was issued and the time any I/O operation already in progress has completed. If the module queue has not been successfully purged, then the WAITC error branch will be taken.

If an uncorrectable hardware error occurs for the operation currently in progress, or if the device is not operational, then the hardware error indicator (CXSGHE) in the ECB gross error indicator byte will be set and the WAITC error branch will be taken.

**Note:** The entry issuing the TPINC PURGE can set the tape name in the tape status table to YYY to prevent normal queuing of I/O requests for the device, but this will not inhibit queuing of I/O requests from macros using a tape name of XXX with auxiliary queue bypass. Any I/O requests which are queued after the TPINC PURGE is issued must be correctly processed, that is, if the tape is auxiliary queued when the TPINC PURGE is issued, then the tape queue must be restarted using TPINC RESTART.

**Restart**

Following normal completion of the I/O operation already in progress (or immediately if there is none), the auxiliary queued indicator in the Tape Status Table will be set off and the ECB I/O counter will be decremented. No error indicators will be set and the WAITC error branch will not be taken.

If an uncorrectable hardware error occurs for the operation currently in progress or if the device is not operational, then the hardware error indicator (CXSGHE) in the ECB gross error indicator byte will be set and the WAITC error branch will be taken.

**Quiesce**

For NORMAL or OFFLINE status, CE1TIN will be set to X'FF' (no auxiliary queue bypass). For AUXQ status, CE1TIN will be set to X'00' (auxiliary queue bypass).

For OFFLINE status the offline indicator in the tape status table will be set, preventing any further queuing of normal I/O requests for this module. However, that this will not inhibit I/O requests from auxiliary queue bypass macros using a tape name XXX. For AUXQ status the auxiliary queued indicator in the tape status table will be set. For NORMAL status the tape indicators will not be modified.

Following completion of all I/O requests (or, in the case of AUXQ status, following completion of all auxiliary queue bypass I/O requests) that were queued at the time the TPINC QUIESCE was issued, the ECB I/O counter will be decremented. No error indicators will be set.

The module queue is considered to have been successfully quiesced if no I/O requests have been added to the queue between the time the TPINC QUIESCE was issued and the time any I/O operation already in progress has completed. If the module queue has not been successfully quiesced, then the WAITC error branch will be taken.

If an uncorrectable hardware error occurs for any operation that was queued at the time the macro was issued or if the device is not operational, then the hardware error indicator (CSXGHE) in the ECB gross error indicator byte will be set and the WAITC error branch will be taken.

**Switch**

The status of the standby tape is checked. If a standby is not available or the status is found to be unsafe, a search is made for a suitable ALT tape. If an ALT tape is found, an attempt will be made to convert the ALT tape to a standby tape.

The available standby tape will become the active tape and I/O requests queued for the old active tape will now be queued for the new active tape. If no errors were encountered during the switch, processing continues at the next sequential instruction (NSI) with X'00' in R15. The ECB may then exit. The switch will continue, asynchronously.

If a standby tape could not be found and it was not possible to convert an ALT tape to a standby, processing continues at the next sequential instruction (NSI) with X'04' in R15.

If a TPINC SWITCH request is issued while there is DDR recovery in progress for the tape specified, processing continues at the next sequential instruction with X'08' in R15.

### ***Mount***

The status of the tape is checked. If there is a tape mounted on that device already, or if the tape status table entry has not been seized, then R15 will contain X'04' on return from the macro and no mount will be performed.

If there is not a tape already mounted, then X'00' will be returned in R15 and mount processing for that device will begin to mount an ALT tape. An ECB will be obtained and control transferred to the tape mount processing segments to complete the request. The final outcome of the request is unknown.

### ***Findalt***

The TLMR entry for the tape mounted on the device specified will be used to locate a suitable ALT tape to switch to.

If a suitable ALT tape was found, the module number will be returned in R14 and the TSTB section 1 address will be returned in R15. The TSTB2 entry will also be seized.

If no suitable tape was found, then R14 will contain X'00'.

### ***Activate***

This function will activate an ECB, currently on the tape wait-list. It begins by scanning the ECBs on the tape wait-list. If an ECB is found waiting for the tape name specified in EBTTNM, it is removed from the tape wait-list. An \$ADPC macro is issued placing the ECB on the ready-list of the I-stream which originally placed it on the tape wait-list.

### ***Getbuf***

This function gets eight 4 KB blocks to make up the blocking buffer and saves the block addresses in the tape status table as well as in the buffer block table.

### ***Relbuf***

This function deletes the addresses of the eight 4 KB blocks for the block buffer from the block buffer table and the tape status table and returns them to the TPF system.

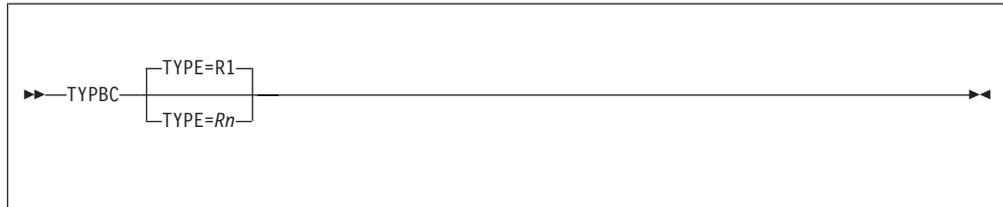
## **Examples**

None.

## TYPBC—Obtain Block Type and Size

Use this system macro to obtain the logical block type and logical size of a storage block in a format suitable for storing in a core block reference word (CBRW).

### Format



#### TYPE=R1|Rn

This register contains the logical storage block type equate value. R0 can not be assigned. The default is R1. R0 is not valid for all calls and R8 through R13 are not valid for E-type calls.

On return the block type and logical size are in this register. Supported types are:

- L0** 128-byte block
- L1** 381-byte block
- L2** 1055-byte block
- L4** 4095-byte block.

### Entry Requirements

The specified register must contain a valid block type.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register specified in the TYPE parameter contains the logical block type and the logical size of the storage block in a format suitable for storing in a CBRW.

### Programming Considerations

- This macro may be processed on any I-stream.
- The program invoking this macro is not required to be in a privileged mode of operation. It is not required to change the protect key, since the instructions used in the inquiry do not modify storage outside the routine's protect key.
- A system error dump can occur when servicing a TYPBC request. See *Messages (System Error and Offline)* for more information.

### Examples

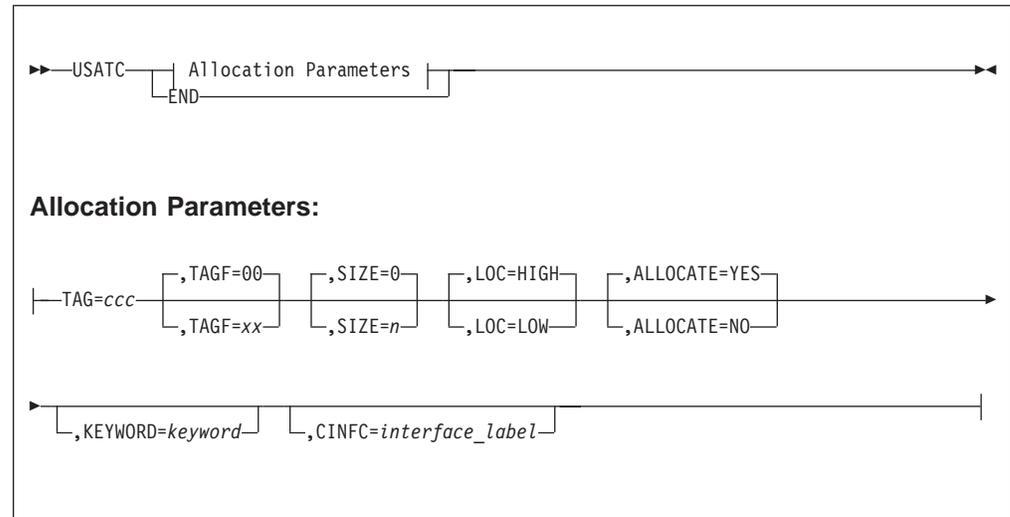
None.

## USATC—Create User Storage Allocation Table Entry

Use this system macro to create an entry in the user storage allocation table (USAT). The system initializer (CT26) uses the contents of the USAT to carve user space in main storage. The USAT information is used only when the CCCTIN user exit is active. The table is mapped by the DSECT IUSAT.

See *TPF System Installation Support Reference* for more information about the CCCTIN user exit.

### Format



#### **TAG=ccc**

Specifies the 3-character dump tag used to identify this area of storage. This parameter is required.

#### **TAGF=xx**

Specifies the 2-hex digit format indicator. This parameter is optional. The default is 00. The format indicators are described in the code for the DLTEC macro.

#### **SIZE=n**

The size, in number of bytes, of the area to be allocated. If size is set to zero, ALLOCATE is set to NO. The parameter is optional and the default is 0.

The area is allocated by CT26 so that it starts and ends on a 4 KB boundary. Aligning the area on 4 KB gives users the ability to key protect their tables.

#### **LOC**

The location where storage is allocated.

##### **HIGH**

Indicates that the area resides in storage above 16 MB.

##### **LOW**

Indicates the area resides in storage below 16 MB.

The default is 31BIT.

#### **ALLOCATE=YES|NO**

Indicates whether or not the area is to be allocated. ALLOCATE=NO can be used to define spare USAT slots. The default is YES.

## USATC

Spare table slots allow new user areas to be created by manipulating the table size using ZAPGM and then reinitializing the TPF system.

**KEYWORD**=*keyword*

Specifies the keyword used to identify the area of storage on the IDOTB macro and the ZIDOT command. The default is no keyword.

**CINFC**=*interface\_label*

Specifies the label used to identify the associated CINFC entry. The default is no label. The label must be defined in UCNFEQ.

**END**

Specifies the end of the USAT build sequence. When specified, this parameter is the only one invoked; other parameters are ignored.

**Note:** If the END parameter is not coded, the TPF system interprets material following proper table entries as though they were also table entries. This results in unpredictable behavior.

## Entry Requirements

None.

## Return Conditions

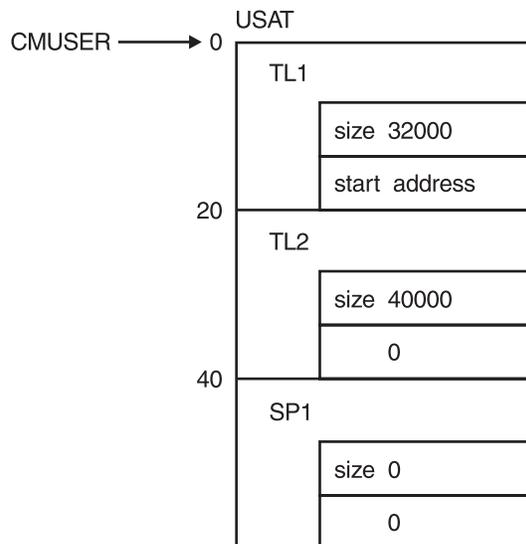
Control is returned to the next sequential instruction (NSI).

## Programming Considerations

- Inline code is generated.
- The CINFC tag CMUSER points at the beginning of the table.
- This macro can be run on any I-stream.

## Examples

The following example shows three entries being built in an allocation table. The table created by the three USATC calls is represented by the following.



1. The following call defines and allocates 32 MB of user storage below 16 MB with a dump tag (TL1), a IDOTB keyword (MYTBL), and a CINFC keyword (UMMYTBL).

USATC TAG=TL1,	Dump tag TL1
SIZE=32000,	32K user area
LOC=LOW,	Allocated below the 16MB
KEYWORD=MYTBL,	IDOTB keyword is MYTBL
CINFC=UMMYTBL,	Reference label is UMMYTBL

2. The next call defines a USAT slot above 16 MB but without allocating it. This is a *dummy* slot. Such a slot might be used during development that is not ready to be activated.

USATC TAG=TL2,	Dump tag TL2
SIZE=40000,	40K user area
LOC=HIGH,	Defined above 16MB
ALLOCATE=NO,	But the area is not allocated
KEYWORD=HITBL1,	IDOTB keyword is HITBL1
CINFC=UMHITBL1	Label is UMHITBL1

3. This allocates a spare entry with SP1 as the its tag. When SIZE=0 (the default), the ALLOCATE parameter is NO.

USATC TAG=SP1	Defines a spare entry
---------------	-----------------------

4. This ends a series of USATC specifications. An END parameter must conclude the USATC calls.

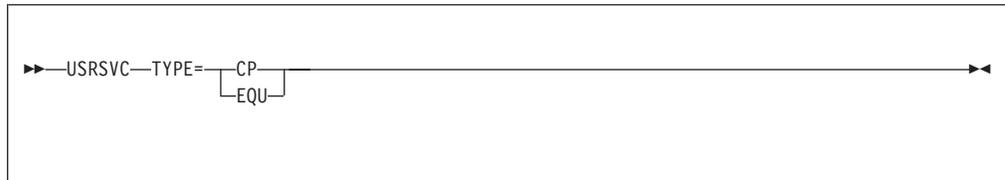
USATC END	Indicates the end of the USAT build
-----------	-------------------------------------

## USRSVC—Generate the User SVC Tables

Use this system macro to create the user indexed supervisor call (SVC) table and the user entries for the primary SVC table. This macro contains the CRESVC macro calls but you can modify the macro to contain the new SVCs.

See “CRESVC—Create an SVC/Fast-Link Table Entry” on page 180 for more information about the CRESVC macro.

### Format



#### TYPE

This parameter specifies whether tables or equates should be generated.

#### CP

CP is used for the table in CCMCDC, and creates the actual primary, secondary, and fast link tables.

#### EQU

EQU creates a set of equates for use in CPSEQ and other areas.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- TYPE=CP should only be used in CCMCDC.
- TYPE=EQU should only be used in CPSEQ.

### Examples

The following set of CRESVC macro invocations define various user SVCs.

- Sample User defined Primary SVCs (USER, USER\_V)

```
CRESVC MACRO=USER1,NUMBER=97,PARMTYP=U,GRPCODE=00,
ROUTINE=A(CPMCIT-CPMAAA),SVCTYPE=USER
```

\*

```
CRESVC MACRO=USER2,NUMBER=127,PARMTYP=U,GRPCODE=00,
ROUTINE=A(CPMCIT-CPMAAA),SVCTYPE=USER
```

\*

```
CRESVC MACRO=USER7,NUMBER=253,PARMTYP=U,GRPCODE=00,
ROUTINE=A(CPMCIT-CPMAAA),SVCTYPE=USER
```

\*

- Sample User defined Indexed SVCs (USER\_I)

\*

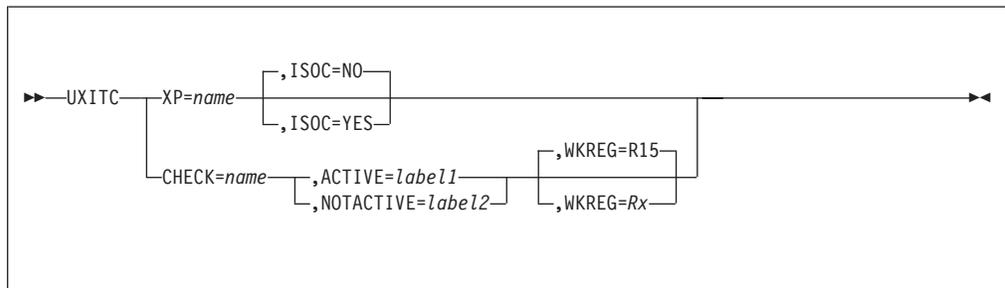
```
CRESVC MACRO=USERI1,NUMBER=000,PARMTYP=F,GRPCODE=00,
ROUTINE=A(CPMCIT-CPMAAA),ECBSYS=(CE1FA0,CE1SUG+1-CE1FA0),
PARMLN=10,SVCTYPE=USER_I
```

```
*  
CRESVC MACRO=USERI2,NUMBER=001,PARMTYP=F,GRPCODE=00,  
ROUTINE=A(CPMCIT-CPMAAA),ECBSYS=(CE1FA0,CE1SUG+1-CE1FA0),  
PARMLN=10,SVCTYPE=USER_I  
*
```

## UXITC–User Exit Interface Linkage

Use this system macro to generate the linkage to a user exit routine from a user exit point (UEP) in the control program (CP).

### Format



#### **XP=name**

This is the name of the user exit, as many as 4 characters. It must satisfy the requirements described in the DCTUCL DSECT. To add an exit point, its name must be added to the exit point list array (UEP) and corresponding entries must be added to the exit point status array (ATT) and to the exit point description array (DESC).

#### **ISOC**

Specifies whether the user exit point is being called from an ISO-C environment.

#### **YES**

TPF ISO-C register conventions must be set up by the user. In particular R13 should be the address of the ISO-C stack.

#### **NO**

R9 is restored upon return from the exit. NO is the default.

#### **CHECK=name**

This is the name of the user exit, as many as 4 characters, to be checked to determine if the user exit is active.

#### **ACTIVE=label1**

This is a label to branch to if the specified user exit is active.

#### **NOTACTIVE=label2**

This is a label to branch to if the specified user exit is not active.

#### **WKREG=R<sub>x</sub>**

This is a register that CHECK parameter processing uses as a work register when one is required.

## Entry Requirements

None.

## Return Conditions

None.

## Programming Considerations

- This macro is only used in the control program (CP) and ISO-C startup routines at defined user exit points. If called from ISO-C startup routines, register R9 must be saved before to the call.

**Note:** CHECK parameter processing can be used by E-type programs to check the status of a user exit.

- I-stream considerations do not apply to the UXITC macro.

## Examples

The following describes how to check the status of a user exit and call it.

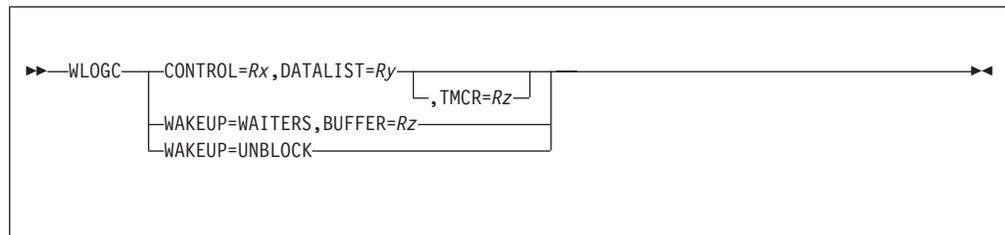
```
UXITC CHECK=SVC,NOTACTIVE=SKIPEXIT
UXITC XP=SVC
SKIPEXIT DS    0H
```

## WLOGC—Write to the Recovery Log

Use this system macro to permit the transaction manager (TM) and resource managers (RMs) to control the writing of information to the recovery log. The recovery log is used by the TPF system in association with TPF transaction services. Recovery log support is general in nature and is designed to be extended to support the recovery of additional resources such as message and tape processing. Application-specific resources can also use the recovery log to become part of the recoverable resources.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### CONTROL=R<sub>x</sub>

A register (R1–R6, R14) that contains the address of the control or header information that is part of the transaction log record. This parameter is required.

#### DATALIST=R<sub>y</sub>

A register (R1–R6, R14) that contains the address of a list of data elements that will be written to the transaction log. This parameter is required.

#### TMCR=R<sub>z</sub>

A register (R1–R6, R14) that contains a pointer to the transaction manager control record (TMCR) associated with this write request. This parameter is optional and, if not coded, will cause the WLOGC service to calculate the address of the TMCR.

#### WAKEUP

This is a special parameter whose use is restricted to the CL22 program. This parameter permits the CL22 program to link to the WLOGC service routine to reactivate resources that are waiting. The following resources are processed:

##### WAITERS

Wake up WLOGC requesters, which are waiting for buffer input/output (I/O) to be completed.

##### UNBLOCK

Wake up WLOGC requesters, which are waiting for WLOGC processing to continue. WLOGC processing is blocked when all buffers are in use.

#### BUFFER=R<sub>z</sub>

A register (R1–R6, R14) that contains a pointer to the buffer associated with the WAKEUP request. This parameter is required when you code WAKEUP=WAITERS.

## Entry Requirements

- For control program (CP) calls, R9 must point to the ECB that is being processed unless you coded the TMCR parameter.
- The control area is defined by ILHDR(ICRCT) and requires that the caller initiate the following fields.

- For Both the ECB-Controlled and CP Program Calls:

Field	Description												
ILH_FLAG	A flag field that identifies the set of WLOGC requests associated with the commit scope and requests services from the log manager. The following indicators are required.												
	<table> <thead> <tr> <th>Indicator</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ILH_FWR</td> <td>First WLOGC request for the commit scope.</td> </tr> <tr> <td>ILH_LWR</td> <td>Last WLOGC request for the commit scope.</td> </tr> <tr> <td></td> <td>Single requests must have both ILH_FWR and ILH_LWR set on.</td> </tr> <tr> <td></td> <td>Middle requests must have both ILH_FWR and ILH_LWR set off.</td> </tr> <tr> <td>ILH_WAIT</td> <td>Indicates that the log manager is to flush the current log buffers out to the recovery log and that the caller wants to wait until all I/O is completed successfully.</td> </tr> </tbody> </table>	Indicator	Description	ILH_FWR	First WLOGC request for the commit scope.	ILH_LWR	Last WLOGC request for the commit scope.		Single requests must have both ILH_FWR and ILH_LWR set on.		Middle requests must have both ILH_FWR and ILH_LWR set off.	ILH_WAIT	Indicates that the log manager is to flush the current log buffers out to the recovery log and that the caller wants to wait until all I/O is completed successfully.
Indicator	Description												
ILH_FWR	First WLOGC request for the commit scope.												
ILH_LWR	Last WLOGC request for the commit scope.												
	Single requests must have both ILH_FWR and ILH_LWR set on.												
	Middle requests must have both ILH_FWR and ILH_LWR set off.												
ILH_WAIT	Indicates that the log manager is to flush the current log buffers out to the recovery log and that the caller wants to wait until all I/O is completed successfully.												
ILH_XID	Commit scope or global transaction ID.												
ILH_PROG	The name of the TM or RM program that will be activated during log recovery.												

- For the CP Program Calls:

Field	Description
ILH_PIA	The postinterrupt address of the RM routine to get control when WLOGC becomes unblocked.

- For the CP Program Calls With ILH\_WAIT Indicated:

Field	Description
ILH_PIA2	The postinterrupt address of the RM routine to get control when WLOGC I/O is completed successfully.
ILH_TOK	A 4-byte token that will be returned to the postinterrupt routine, which is defined by ILH_PIA2, on activation.
ILH_PIN	An index value (processor ordinal number X 4) for the processor to write to.
ILH_ISN	The I-stream number on which the postinterrupt routine gets control.

- The data list is defined by ILRDA(ICRCT) and requires the caller to initialize the following fields.

Field	Description
ILR_CNT	Count of data items to log with this request. Multiple data items may exist, giving the RM the ability to log multiple pieces of a data with one WLOGC call.

## WLOGC

ILR_LEN	The length of the data item to log.
ILR_LOC	The location of the data item to log.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R0–R7 are preserved across this macro call.
- The condition code (CC) is not preserved across the macro call.
- For CP program calls, the macro service routine sets R14 to contain a hexadecimal return code.

Return Code	Meaning
-------------	---------

R14=0	The recovery log write has been completed successfully.
R14=-1	The request was rejected; the WLOGC macro is blocked because of an out-of-buffer condition.

## Programming Considerations

- This macro can be run from any I-stream.
- This macro can be run by both ECB-controlled programs and CP programs.
- The amount of data that can be written with one WLOGC request is limited to 32 KB.
- The caller loses control when ILH\_WAIT is indicated; it will not be reactivated until I/O is completed successfully on the associated buffer and all buffers before it.

Reactivation differs depending on the residency of the caller.

- When the caller is E-type code, control is returned at the next sequential instruction (NSI) after the supervisor call (SVC).
- When the caller is CP code, control is returned to ILH\_PIA2 on the indicated I-stream with the value of ILH\_TOK passed back.
  - R1 contains the address of a system work block (SWB) with ILH\_TOK stored in location 0. The SWB must be released by the post interrupt routine.
  - R2 contains ILH\_PIA2, the address of the post interrupt routine.
- Processing can be delayed if a WLOGC request is issued and there are no buffers available (WLOGC processing is blocked). Processing differs depending on the residency of the caller:
  - When the caller is E-type code, the ECB is forced to wait and the WLOGC SVC is issued again. The ECB waits until WLOGC processing is unblocked.
  - When the caller is CP code, the WLOGC request is rejected and the WLOGC blocked return code is returned. The caller must suspend processing until it receives a wakeup call from the log manager. The log manager saves the passed postinterrupt address and wakes up the caller when WLOGC processing becomes unblocked.

Return Code	Meaning
-------------	---------

R1	Contains the address of a system work block (SWB) that must be released by the post interrupt routine.
R2	Contains ILH_PIA, the address of the post interrupt routine.

## Examples

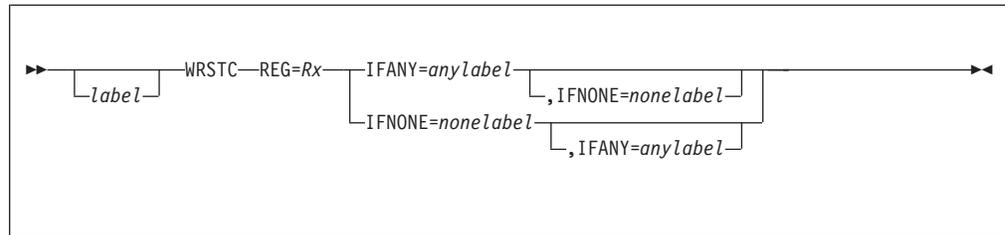
The following is an example of how to write to the recovery log.  
WLOGC CONTROL=R2,DATALIST=R3

## WRSTC–Get Load Module Writable Static Data Length

Use this system macro to get the writable static data length from a load module's IDSLST.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

### Format



#### REG=*Rx*

A register other than R0 in which the length of the writable static data is returned. This is a required input parameter.

#### IFANY=*anylabel*

The symbolic name of a label to which control is transferred if writable static data is present.

#### IFNONE=*nonelabel*

The symbolic name of a label to which control is transferred if writable static data is not present.

### Entry Requirements

- A base register must address the IDSLST of the load module for which the writable static data length is required.

### Return Conditions

- If the load module contains writable static data:
  - If IFANY=*anylabel* is specified, control branches to *anylabel*.
  - If IFANY=*anylabel* is not specified, control is returned to the next sequential instruction (NSI).
- If the load module does not contain writable static data:
  - If IFNONE=*nonelabel* is specified, control branches to *nonelabel*.
  - If IFNONE=*nonelabel* is not specified, control is returned to the next sequential instruction (NSI).
- Register *Rx* contains the writable static data length. If there is no writable static data, *Rx* contains zero.
- The contents of all other registers are preserved across the macro call.
- The condition code (CC) is not preserved across the macro call.

### Programming Considerations

- This macro can be run from any I-stream.

- This macro can be run by both ECB-controlled programs and control program (CP) programs.

## Examples

The following example shows how to get a load module's writable static data length.

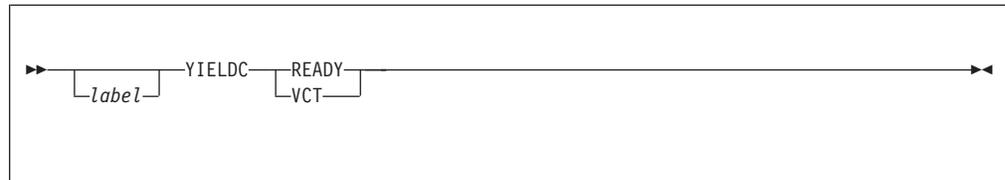
```
*****
* Register R4 points to the beginning of the load module.          *
*****
        IDSLST REG=R4
        WRSTC REG=R15,IFNONE=NOSTATIC
*****
* If control falls through to here, R15 contains the writable static *
* data length.                                                       *
*****
        NOSTATIC DS    0H
*****
* If control branches to here, there is no writable static data. R15 *
* contains zero.                                                     *
*****
```

## YIELDC–Yield Control

Use this system macro to yield control of the processor and allow processing of other entries. The entry is placed on the specified processor list.

Required Authorizations			
Key0	Restricted	System	Common Storage
	X		

## Format



### *label*

A symbolic name assigned to the macro statement.

### **READY**

Assigns the entry to the *ready list*. This is the highest priority list available to which the entry can be assigned. Assigning the entry to this list prevents any new work from being processed by this I-stream before the entry is dispatched again. Any entries that are already on the ready list will be allowed to process.

### **VCT**

Assigns the entry to the *virtual file access count (VCT) list*. This list is used to delay the processing of an entry until all work on the ready list has been processed. Any entry on the VCT list is interleaved with new work. Entries on the VCT list will still be processed even when the system has stopped processing new work because a shutdown condition (low resource condition) has occurred.

## Entry Requirements

R9 must contain the address of the entry control block (ECB) that is being processed.

## Return Conditions

- Control returns to the next sequential instruction (NSI).
- The contents of register 14 (R14) and register 15 (R15) are unknown. The contents of all other registers are preserved across this macro call.
- The condition code is not saved across this macro call.

## Programming Considerations

- Use of this macro should be limited; excessive use macro will cause a low-core condition in the real-time system.
- This macro can be run on any I-stream.
- After this macro runs, the ECB is added to the specified CPU list. Accordingly, control can be transferred for processing of another entry.

- Records must not be held by the ECB when the YIELDC macro is issued. If the entry that is issuing YIELDC is holding a record, a number of entries (also holding storage blocks) can be queued for that record. This can lead to a lockout condition.
- When you run this macro, the 500-millisecond program timeout is reset.
- The macro trace collection for YIELDC compresses multiple occurrences of YIELDC into two entries. This prevents programs that are issuing successive YIELDC macros from filling up the macro trace table by keeping only the first and last YIELDC trace information. In addition to the normal macro trace information, the first YIELDC trace entry contains a count of suppressed entries.
- When it is run during system cycle down, this macro is handled like a DEFRC macro by the TPF system and the entry is placed on the *defer list*.

## Examples

1. You call YIELDC in order to add the ECB thread to the VCT list.

```
YIELDC VCT
```

The output from YIELDC VCT will look similar to the following:

```
YIELDC VCT
00CBB000 *           YIELDC QPN2 +0AA
050C0000 003880AC 003B 0026 AE405F11 6760BE09
```

2. You call YIELDC in order to add the ECB thread to the ready list

```
YIELDC READY
```

The output from YIELDC READY will look similar to the following:

```
YIELDC READY
006D9000 *           YIELDC QPN2 +1CA
050C0000 003891CC 003B 0026 AE405F1B D887CA00
```

3. You call YIELDC in order to add the ECB thread to the VCT list; it loops 5 times:

```
YIELDC VCT
```

The output from YIELDC VCT will look similar to the following:

```
YIELDC VCT
0072A000 *           YIELDC QPN2 +18E      000004 ENTRIES SUPPRESSED
050C0000 0038A190 003B 0026 AE405F2B 5414AE02
```

4. You call YIELDC in order to add the ECB thread to the ready list; it loops twice:

```
YIELDC READY
```

The output from YIELDC READY will look similar to the following:

```
YIELDC READY
004054A0      IS01 YIELDC QPN2 +1AA      000001 ENTRIES SUPPRESSED
050C2000 0038C1AC 003B 0026 AE41633E AC6F9C08
```

**YIELDC**

---

# Index

## Special Characters

\$ADPC macro 12  
\$CKMAC macro 14  
\$CONBC macro 16  
\$CPUC macro 19  
\$CRISC macro 22  
\$DCOLC macro 24  
\$DISBC macro 27  
\$FINDC macro 29  
\$FORKC macro 32  
\$GCOMC macro 33  
\$GETBC macro 35  
\$GETRC macro 38  
\$GEVAC macro 40  
\$GIOBC macro 42  
\$GMNBC macro 44  
\$GSVAC macro 46  
\$GSWBC macro 48  
\$GSYSC macro 50  
\$GTSTC macro 53  
\$LCKRC 55  
\$LOCKC macro 56  
\$MASKC macro 58  
\$MONTC macro 61  
\$MOVEC macro 62  
\$RCOMC macro 64  
\$REVC macro 66  
\$RELBC macro 68  
\$RELRC macro 70  
\$RETRC macro 72  
\$RIOBC macro 75  
\$RMNBC macro 77  
\$RSWBC macro 79  
\$RSYSC macro 81  
\$SWSPC macro 84  
\$TCPLC macro 87  
\$ULKRC 90  
\$UNLKC macro 91  
\$VALEC macro 93  
#IPxxx 232  
#SBRC macro 94

## Numerics

3705 Communications macro 403

### A

access recoup descriptor record for recoup 264  
Access RIAT Entry macro 407  
Access the Record Cache Subsystem Status Table macro 400  
access the subsystem status table (SSST) 400  
acquired storage  
  release 77, 81  
Activate or Deactivate C Function Trace for an ECB macro 221

Add a Block to Top of a Dispatch List macro 98  
Add an Entry to the Sense Table macro 328  
Add Block to the End of a Dispatch List macro 101  
Add Work to a List on Specified I-stream macro 12  
ADDFC macro 98  
ADDLC macro 101  
address  
  conversion 40  
  conversion between EVM and SVM 276  
  conversion of EVM to SVM 46  
  for RNHPT entry 212  
  get block 48  
  release I/O control block 75  
  space switch 84  
  validation for EVM 93  
allocation  
  create storage table 501  
  of user storage 501  
application timeout 428  
area deactivation switch  
  capture switch 373  
  update switch 373  
asynchronous I/O 392  
ATA/IATA format telegram 422  
authorization  
  for macro use 14  
  MOVEC 351  
  protect key zero  
  MOVEC 350  
authorize macro use 14

### B

BACKC macro  
  data collection for 24  
BBEWP macro 104  
BBWRT macro 108  
BKDx record initialization 306  
block  
  address  
    get available I/O 42  
    get size 376  
    getting a CLAW block 255  
    getting a coupling facility block 254  
    release storage 64  
    release system work 79  
    releasing a CLAW block 397  
    releasing a coupling facility block 396  
    return storage 68  
    type and size  
      get 500  
block address  
  get 48  
block types supported 500  
BRSTR macro 113  
BSAVE macro 115  
BSC message 422  
BSCQC macro 117

BSTAK macro 118  
buffers  
  flush a record from VFA 241, 246  
Build Selective Memory Dump Table macro 289  
BXDx records 264

## C

C function trace 221  
C function trace information for an ECB 431  
Call a Secondary Library Routine macro 285  
CCSONS DASD services 233  
CEBIC macro 122  
CF asynchronous queue 218  
CFCONC macro 128  
CFDISC macro 143  
CFISVC macro 146  
CFLF asynchronous queue 218  
CFRQC macro 148  
CFVCTC macro 152  
Chain Chase  
  example 270, 306  
  scheduler 268  
  through Prefix Pages macro 366  
Chain Chase through Prefix Pages macro 366  
Change Addressing Mode macro 347  
Change MDBF Subsystem/Subsystem User ID  
  macro 122  
Change Protection Key macro 332  
Change the System Mask macro 58  
change to another i-stream 467  
channel keypoint setup 189  
characteristics of file addresses 226  
check a list notification vector 152  
Check or Modify a List Notification Vector macro 152  
Check Symbolic Line Type macro 383  
CIO 323  
CIO I/O processing 405  
CIO processing suspend 457  
CIO resume 405  
CIOSC macro 159  
CIOUC macro 161  
CLAW API Linkage macro 164  
CLAW block type 255  
CLAW device support  
  API linkage 164  
  code generation 164  
  get CLAW block type 255  
  release CLAW block type 397  
  write a CLAW error log 477  
CLAW trace record 477  
CLAWCC macro 164  
Clean Up Blocks in the CRET macro 399  
CLH  
  ADDLC 101  
  NUMLC 361  
  NXTLC 363  
CLNKC macro 166  
CMB authorization 14  
code generation for CLAW API 164  
Common Block authorization 14

common I/O processing suspend 457  
Common Link to Access Workstation (CLAW) 164  
common storage block  
  get 33  
  release 64  
communicate with processors 435  
communication  
  3705 403  
  between processors 437  
  data 381  
  line management 161  
  line status 386  
Compute File Address macro 228  
CONBC  
  See \$CONBC  
Connect Block to ECB Virtual Memory macro 16  
Connect to a Coupling Facility List or Cache Structure  
  macro 128  
contiguous EVM storage  
  get 44  
control block area lock 304  
Control MPIO Device macro 358  
Control Program (CP) Call and Link macro 166  
Control Program (CP) Rounding macro 172  
Control Program (CP) Save Link Data and Set Stack  
  Pointer macro 448  
Control Program (CP) Tape Logging macro 87  
Control Program Linkage 1  
control record modification 72  
control record retrieval 38, 72  
Control the TPF System MP Environment macro 369  
control transfer 204  
convert  
  EVM address to SVM address 46  
  ITSTB pointer 197  
  SVM address to EVM address 40  
Convert an EVM Address to an SVM Address  
  macro 276  
Convert EVM Address to SVM Address macro 46  
Convert SVM Address to EVM Address macro 40  
Convert Tape Status Table Pointer macro 197  
core blocks  
  releasing 117  
count queued blocks 361  
coupling facility  
  getting a work block address 254  
  releasing a work block address 396  
Coupling Facility Request macro 148  
CP calling routine return 409  
CP dsect 168  
CP Find a File Record macro 29  
CP linkage 166  
CP subroutine interface 210  
CPDSC macro 168  
CPLKC macro 169  
CPRND macro 172  
CRAS - get file pool address switch 373  
CRAS status table 175  
CRASC macro 173  
CRATC macro 175

- create
  - ECB 204, 467
  - user storage allocation table 501
- create a macro group definition 179
- Create a New ECB and Transfer Control macro 204
- Create an Asynchronous ECB macro 32
- Create an SVC/Fast-Link Table Entry macro 180
- Create User Storage Allocation Table macro 501
- CREGPC macro 179
- CRE SVC macro 180
- critical message macro 200
- CROSC macro 185
- Cross Over to Another I-stream macro 22
- Cross-Subsystem Access Service Request macro 185
- CT26 initialization 501
- CTKL macro 189
- CVTPC macro 197
- CWRTC macro 200
- CXELTY tags 384
- CXFRC macro 204
- Cycle Down Utility CP Interface macro 210
- CYDNC macro 210

## D

- DASD services 233
- DAT mode 58
- data
  - collection
    - control with ZMEAS 24
    - hook insertion macro 24
    - control during recoup 264
    - control for CP 448
    - movement between EVM and SVM 62, 350
    - purge from queue 325
    - transfer
      - tape 478, 481
  - data chain transfer
    - file 237
  - Data Collection Hook Insertion macro 24
  - data definition names for GDS
    - locating 258
  - data event control block (DECB) 29
  - DECB DSECT 29
  - define a macro group 179
  - Define and Enqueue Resource, Signal Aware macro 419
  - Define Stack DSECT for Control Program (CP) Routine macro 216
  - descriptor record access for recoup 264
  - device handler 323
  - DHASHC macro 212
  - diagrams for macro models xv
  - disable C function trace 221
  - DISBC macro
    - See \$DISBC macro
  - Disconnect A Block from the ECB Virtual Memory macro 27
  - Disconnect from a Coupling Facility List or Cache Structure macro 143

- Dismount a Symbolic Device Address (SDA) macro 217
- dismount an SDA 159
- dismount tape 495
- dispatch
  - task 12, 98
- dispatch list
  - count blocks queued 361
  - management 101
- dispatch suspended ECB 219
- display file pool counts switch 373
- DLCKC macro 214
- DLNKC macro 216
- DSDAC macro 217
- dump
  - override table
    - building 296
    - table 289
- Dump Override Table Build macro 296
- dynamic address translation mode 58

## E

- ECB
  - suspend for I/O 349
  - trace information 431
  - virtual memory 84
    - connecting 16
    - disconnect block 27
- ECBLC macro 218
- ELLEC macro 219
- enable C function trace 221
- Enable/Disable Program Event Recording (PER) macro 371
- ENATC macro 221
- entry
  - in the macro information tables 146
- ENTxC macros
  - data collection for 24
- error
  - log for CLAW
    - TCLAC macro 477
- ESFAC macro 226
- events
  - asynchronous I/O 392
- EVM 84
- EVM Address 40
- EVM address conversion 276
- EVM address to SVM address conversion 46
- EVM address validation 93
- EVM storage
  - get contiguous 44
- EVM to SVM data movement 62, 350
- Example of an IBM indexed SVC: 184
- exit interface linkage 506
- External interrupts 58

## F

- FACE interface 228
- FACS interface 228

- FACZC macro 228
- fast link macros in macro decoder 146
- fast-link macro table
  - creation 180
- FCTLC macro 233
- FDCTC macro 237
- file
  - storage activity data collection 24
- File a Special Record macro 243
- file address
  - characteristics 226
  - computation 228
  - computation for fixed 264
  - information
    - getting 218, 226
- File Control macro 233
- File Data Chain Transfer macro 237
- file macro
  - data collection 24
- file record find 29
- Find a Special Record macro 249
- Find Entry in the Macro Information Tables macro 146
- find macro
  - data collection 24
- Find SLST Entry macro 386
- Find/File macro 252
- fixed file address computation 264
- FLFAC macro 241
- FLSPC macro 243
- Flush a Record from VFA Buffers macro 241, 246
- FLVFC 246
- FNSPC macro 249
- FTSTC macro 252
- function trace for C 221
- Functional Areas
  - Area 03B – Recoup
    - BBEWP 104
    - BBWRT 108
    - GROUP 264
    - INDEX 306
  - Area 21A – User Exits
    - UXITC 506

## G

- GCFBC macro 254
- GCLAC macro 255
- GCOMC macro
  - See \$GCOMC macro
- GDSCC macro 258
- General Data Set (GDS) Control macro 258
- General Tape Data (GDS)Transfer macro 481
- General Tape Data Chain Transfer macro 478
- Generate Control Program (CP) DSECT macro 168
- Generate IBM SVC and Fast-Link Tables macro 282
- Generate Selective Memory Dump Table Entry macro 293
- Generate the User SVC Tables macro 504
- Get a Control Record macro 38
- Get a Specified CLAW Block Type macro 255
- Get a System Work Block (SWB) Address macro 278

- Get Address of Next Block Queued on a Dispatch List macro 363
- Get Available I/O Control Block Address macro 42
- get common storage block 33
- Get Contiguous EVM Stack Storage macro 53
- Get Contiguous EVM Storage macro 44
- Get Count of Blocks Queued on a Dispatch List macro 361
- Get Coupling Facility Work Block Address macro 254
- Get Global Attribute Table Entry macro 302
- get hash address for resource name 212
- Get Load Module Writable Static Data Length macro 512
- Get Maximum Number of Storage Blocks macro 345
- Get next TPF Trace Table Entry macro 367
- Get Program Prolog Area (PPA) Functional Name Information macro 261
- get record code check (RCC) reference table 275
- Get System Heap Storage macro 50, 279
- Get System Work Block Address macro 48
- get the unique token for the current transaction 492
- GETCC Block Type Verification macro 331
- getting
  - system work block address 48
- GFS Parameter Program switch 373
- GNAMC macro 261
- GROUP macro 264
- GRRTC macro 275
- GSVAC macro 276
- GSWBC macro 278
- GSYSC macro 279

## H

- Halt an I/O Operation macro 281
- Hash Resource Name macro 212
- heap storage
  - getting 50, 279
  - releasing 81, 413
- high speed transmission for LMT 446
- high-performance routing (HPR) support
  - core blocks
    - releasing 117
- high-speed display message 422
- HIOSC macro 281
- hook insertion for data collection 24
- HPR control message
  - sending 453
  - writing 453
- HPR support
  - core blocks
    - releasing 117

## I

- i-stream change 467
- I/O block address
  - get available 42
- I/O completion
  - suspend ECB 349

- I/O control block address
  - release 75
- I/O events 392
- I/O Interrupt Status Update 214
- I/O interrupts 58
- I/O operation
  - resetting 406
- I/O Operation
  - starting 436
- I/O processing 405
- I/O request
  - preemptive 378
  - return from 380
- IBMSVC macro 282
- ICELOG macro 283
- ICLANC macro 285
- ICPLOG macro 286
- ID shift for SS/SSUs 337
- IDATB macro 289
- IDATG macro 293
- IDOTB macro 296
- IDSDEC 29
- IFRVTC macro 300
- IGATC macro 302
- ILCKCB macro 304
- INDEX macro 264, 306
- indexed SVC 180
- Initialize and Reset Communication Lines macro 161
- Initiate A Preemptive I/O Request macro 378
- insert hook for data collection 24
- intercepts for data collection 24
- Interface for SIGP Services macro 19
- interface to data collection 24
- interprocessor communications 437
- Interrogate Symbolic Device Address (SDA) Status macro 326
- interrupt processing for CIO 323
- Interrupt Status Update 214
- Invoke TCP/IP Native Stack Common Service Routine macro 324
- IOBs 218
- IOCB address
  - get 42
  - release 75
- IOIRC macro 323
- IPC 437
- IPC service 435
- IPC Service Request macro 435
- IPSDC macro 324
- IPURGE macro 325
- ISDAC macro 326
- ISNSE macro 328
- ITSTB pointers 197
- IULKCB macro 330
- IVTYPE macro 331

## K

- key 0 61
- Key0 authorization 14
- KEYCC macro 332

- Keypoint Communication Data macro 381
- keypoint setup for SLC channels 189
- Keypoint Update macro 334
- KEYUC macro 334

## L

- LCPCC macro 335
- LEBIC macro 337
- LEMIC macro 340
- line type checking 383
- line type suffixes 384
- link data control for CP 448
- Link to CP Routines macro 169
- linkage
  - example 1
- Linkage 1
  - linkage for CP 409
  - linkage for CP routine 166
  - linkage for user exits 506
  - linkage routine 94
- LLF attention queue 218
- LMONC macro 344
- LMT high speed transmission 446
- Load and Shift SS/SSU ID macro 337
- local terminal message 422
- Lock a Resource macro 56
- lock a VFA exclusive lock 55
- lock a VFA shared lock 55
- Lock a Virtual File Access (VFA) Shared Lock or Exclusive Lock macro 55
- Lock Control Block Area macro 304
- Lock Entry Management Interface macro 340
- lock management 219
- lock release on WGTA entry 418
- Lock resource 214
- locked program exit 219
- Lockword update 214
- log an error for CLAW 477
- long life entry detection 219
- long running ECB 488
- looping program exit 219
- low address protect 335
- Low Address Protect Set and Restore macro 335
- low core dsect 168

## M

- macro decoder table address 146
- macro group definition 179
- macro information table entries 146
- macro model diagrams xv
- macro use authorization 14
- main storage record location 264
- main storage records 264
- MAXBC macro 345
- MDBF SSID/SSUID macro 122
- memory dump table 289
  - generating 293
- message to terminal 422
- message traffic data collection 24

- miscellaneous SON pool functions switch 373
- MODEC macro 347
- models of macro invocations xv
- modify a control record macro 72
- modify a list notification vector 152
- Modify Lock and I/O Interrupt Status macro 214
- MONTC authorization 14
- MONTC macro 348
- MONWC macro 349
- Mount a Symbolic Device Address (SDA) macro 356
- mount an SDA 159
- move data between 2 different EVMS 62
- move data between EVM and SVM 62, 350
- Move Data Between EVM and SVM macro 62, 350
- MOVEC macro 350
- MPIFC macro 352
- MSDAC 356
- MSGIN macro
  - data collection 24
- MSPIC macro 358

## N

- native subchannel communications for 3705 403
- NCB directory record
  - getting ordinal number of 212
- network layer packet (NLP)
  - retransmitting 453
  - sending 453
  - writing 453
- NSC communication for 3705 403
- NUMBC macro 360
- number of storage blocks
  - getting 345
- number of storage blocks available 360
- NUMLC macro 361
- NXTLC macro 363
- NXTPC macro 366
- NXTRC macro 367

## O

- Obtain Block Type and Size macro 500
- Obtain Common Storage Block macro 33
- Obtain Storage Block macro 35
- Obtain Symbolic File Address Information macro 226
- OMT message 462
- ordinal number
  - for NCB directory record 212
- output message queuing 388
- override dump bitmap table macro 296

## P

- path information unit (PIU)
  - sending 453
  - writing 453
- PAUSC macro 369
- PER interrupts 58
- PERCC 371
- perform subsystem function 392

- PFSWC macro 373
- PHYBC macro 376
- PIO 405
- PIOFC macro 378
- PIORC macro 380
- PIU to SNA NCP data 453
- PKEYC macro 381
- Place on Queue macro 388
- PLNAC macro 383
- PLNSC macro 386
- PLONC macro 388
- pointer conversion for ITSTB 197
- pool
  - FARF address generation 231
- pool directory 373
- predefined record type names 232
- preemptive I/O request 378
- preemptive I/O return 380
- prefix page chain chase 366
- presentation services input list queue purge 325
- printer message 422
- problem state 344
- processor communications 437
- PROGC macro 390
- program activity
  - ENTxc/BACKC 24
- program activity data collection 24
- Program Check recovery 66
- program prolog area function name information 261
- protect low core addresses 335
- PSW access 61
- Purge Data from Queue macro 325
- purge tape operations 495

## Q

- QASNC macro 392
- QGDSQ 394
- Query Asynchronous I/O Event Facility macro 392
- Query General Data Set (GDS) Input/Output (I/O) Queue 394
- query general data set device I/O queue macro 394
- Query Number of Storage Blocks Available macro 360
- queue data purge 325
- queue manipulation for SLC 443
- quiesce tape operations 495

## R

- railroad tracks xv
- RCFBC macro 396
- RCLAC macro 397
- RCRTC macro 399
- RCS device notification 392
- RCS status table access 400
- RCSSC macro 400
- RDCTC macro 403
- Read and Process Program Version Record macro 411
- reconcile file pool counts switch 373
- record cache subsystem device notification 392

- record cache subsystem status table 400
- record hold table wait queue 218
- Record ID attribute table access 407
- record type names 232
- records
  - reading non-TPF 249
  - writing non-TPF 243
- Recoup Descriptor Record Access macro 264
- Recoup Descriptor Record Structure macro 306
- Recoup Error Item Setup macro 104
- recoup keypoint for chain-chasing 266
- recoup Logging Item Setup macro 108
- recoup pool function switch 373
- Recoup Register and Work Area Save macro 115
- Recoup Register Restore macro 113
- recoup stack area
  - controlling 118
- Recoup Stack macro 118
- Recover from Program Check macro 66
- Release a Control Record macro 70
- Release a Lock on a WGTA Entry macro 418
- Release a Specified CLAW Block Type macro 397
- Release a System Work Block (SWB) macro 412
- Release Acquired Storage macro 77
- Release Common Storage Block macro 64
- Release Core Blocks That Are Not Attached to an ECB macro 117
- Release Coupling Facility Work Block Address macro 396
- Release Input/Output Control Block (IOCB) Address macro 75
- release storage block 68
- Release Storage Block macro 68
- Release System Heap Storage macro 81, 413
- Release System Work Block macro 79
- releasing
  - I/O control block address 75
- Remove IOBs Associated with an ECB Address macro 218
- Request a Mount, Dismount or Status of an SDA macro 159
- request IPC service 435
- Request MPIF Service macro 352
- Reset an I/O Operation macro 406
- reset communication lines 161
- Reset Pool Function Switch macro 373
- Reset Supervisor State (Problem State) macro 344
- RESMC macro 405
- resource locking 56
- resource name hash prime table (RNHPT) entry
  - obtaining address of 212
- restart tape operations 495
- Restricted authorization 14
- restricted macro authorization 14
- resume a suspended ECB 219
- Resume Normal CIO I/O Processing macro 405
- retrieve a control record macro 72
- Retrieve or Modify a Control Record macro 72
- Return from CIO Input/Output (I/O) Interrupt Processing macro 323
- Return from PIO I/O Interrupt Processing macro 380

- return program information 371
- Return Program Information macro 390
- Return the Physical Size of a Storage Block macro 376
- Return to CP Calling Routine and Reset Stack Pointer macro 409
- RIAT access 407
- RIOBC macro
  - See \$RIOBC macro
- RIOBC macro 406
- RITID macro 407
- RLNKC macro 409
- RNHPT entry
  - obtaining address of 212
- ROUTC macro
  - data collection 24
- RPVRC macro 411
- RSWBC macro 412
- RSYSC macro 413
- RVT entry search 415
- RVTCC macro 415
- RWGTC macro 418

## S

- SANQC macro 419
- save link data and set stack pointer 448
- Schedule an ECB to Exit or Resume macro 219
- SDA 281, 326
- SDA halt and clear I/O 406
- SDA I/O 436
- Search CRAS Status Table macro 175
- Search RVT Entries macro 415
- selective memory dump table
  - building 289
  - generating entries 293
- Send Control Message to 3270 macro 450
- send data to SNA LU 453
- Send LMT High Speed Transmission macro 446
- Send LU 6.2 Message from OMT macro 462
- Send Message to CRAS macro 173
- Send Message to Terminal macro 422
- SENDC macro 422
  - data collection 24
- service for subsystem access 185
- service request for MPIF 352
- Set and Test Lethal Utility Switch macro 464
- Set C Function Trace Information for an ECB macro 431
- Set Maximum Times to Avoid Application Timeout for an ECB macro 428
- set stack pointer and save link 448
- set subsystem mode CCW 392
- Set Supervisor State (Monitor Mode) macro 348
- Set Supervisor State (Monitor Mode) with PSW Return macro 61
- Set System Mask macro 459
- SETOC macro 428
- SETTC macro 431
- shift SS/SSU IDs 337
- SICFC macro 435

- SIGP services interface 19
- SIOSC macro 436
- SIPCC macro 437
  - data collection 24
- size
  - get block 500
- size of storage blocks
  - get 376
- SLC Channel Keypoints Setup macro 189
- SLC message 422
- SLC Queue Handling macro 443
- SLC transmission 388
- SLCQC macro 443
- SLMTC macro 446
  - data collection 24
- SLNKC macro 448
- SLST 383
- SLST entry
  - finding 386
- SNA I/O
  - writing 453
- SNA NCP from PIU data 453
- SNDLC macro 450
- SOUTC macro 453
- Special Tape Interface macro 495
- spin lock 56
- SPNDC macro 457
- SS/SSU ID shift 337
- SSID/SSUID macro 122
- SSMMC macro 459
- SSST access 400
- STA/STP 2314 GFS randomizer switch 373
- stack area use
  - sample 1
- stack definition
  - example 1
- stack pointer reset for calls 409
- stack storage
  - get contiguous EVM 53
- Standard Linkage Macro Subroutine macro 94
- Start an Input/Output (I/O) Operation macro 436
- start task 12, 98
- status of an SDA 159
- STIMC macro 460
- STLUC macro 462
- storage
  - get contiguous EVM 44
  - release acquired 77, 81
  - system heap 50, 81, 279
- storage allocation table creation 501
- storage blocks
  - available 360
  - connecting 16
  - disconnecting 27
  - get 33
  - get size 376
  - getting 345
  - release 64, 68
- subsystem access request 185
- subsystem id/subsystem user id 122
- subsystem status table (SSST) access 400
- supervisor state 61, 344, 348, 459
- supported block types 500
- Suspend ECB, Pending I/O Completion macro 349
- suspend long running ECB 488
- Suspend Normal CIO Processing macro 457
- SVC definition 180
- SVC numbers in macro decoder 146
- SVM 84
  - SVM Address 40
  - SVM address conversion 276
- SWCHC macro 464
- SWISC macro 467
- Switch Address Space macro 84
- Switch Entry to Another I-Stream macro 467
- SYCON macro 471
- symbolic device address (SDA) 281, 436
- symbolic device address (SDA) status 326
- symbolic file address information
  - getting 218, 226
- symbolic line status table 386
- symbolic line type checking 383
- syntax diagrams xv
- System authorization 14
- System Configuration macro 471
- system heap storage
  - getting 50, 279
  - releasing 81, 413
- System Interprocessor/Inter—I—Stream Communication macro 437
- System Macros
  - INDEX 306
  - macros for miscellaneous packages
    - BBEWP 104
    - BBWRT 108
    - GROUP 264
- system mask 459
  - changing 58
- system virtual memory (SVM) 84
- system work block (SWB)
  - getting 278
  - releasing 412
- system work block release 79

## T

- table address for record ID attributes 407
- TANCC macro 473
- Tape Control macro 493
- tape data transfer 478, 481
- tape operation interface 495
- tape switch initiation 495
- TARGET(TPF) C Language Support Epilog macro 283
- TARGET(TPF) C Language Support Prolog macro 286
- TASBC macro 475
- TASTC macro 476
- TCLAC macro 477
- TCP/IP native stack
  - invoking the common service routine 324
- TDCTC macro 478
- TDTAC macro 481
- TERMC macro 485

- terminal message 422
- Test Input/Output (I/O) Service macro 486
- Test RID/RVT address macro 300
- threaded processes
  - ending 485
- Time Slice an ECB macro 488
- Time-Initiated CP Routine Execution macro 460
- timeout 270
- TIOSC macro 486
- TMSLC macro 488
- TMTKC macro 492
- TPCNC macro 493
- TPINC macro 495
- trace information for an ECB 431
- transaction anchor table 473
- Transaction Anchor Table Control (TANC) macro 473
- transfer
  - file data chain 237
- transfer control 204
- Transmission Control Protocol/Internet Protocol (TCP/IP)
  - CLAWCC macro 164
  - GCLAC macro 255
  - RCLAC macro 397
  - TCLAC macro 477
- Turn Off Time Available Supervisor Switch macro 475
- Turn on Time Available Supervisor Switch macro 476
- TYPBC macro 500
- type
  - get block 500
- types of blocks supported 500

## U

- Unlock a Control Block Area macro 330
- unlock a record hold table (RHT) 90
- Unlock a Resource macro 91
- unlock a virtual file access (VFA) 90
- Unlock a Virtual File Access (VFA) or Record Hold Table (RHT) Lock macro 90
- Unlock resource 214
- USATC macro 501
- User Exit Interface Linkage macro 506
- user storage allocation 501
- USRSVC macro 504
- UXITC macro 506

## V

- Validate Entry Control Block (ECB) Virtual Memory (EVM) Address macro 93
- Validate Use of Restricted Macro macro 14
- validation of EVM addresses 93
- VFA 29
- VFA buffers
  - flush a record 241, 246

## W

- WGTA entry lock release 418
- WLOGC 508

- work block
  - release system 79
- work block address
  - get 48
- Write a CLAW Error Log macro 477
- Write Critical Message to the System Console macro 200
- Write Path Information Unit (PIU) Systems Network Architecture (SNA) Input/Output (I/O) macro 453
- Write to the Recovery Log macro 508
- WRSTC 512

## Y

- Yield Processing macro 514
- YIELDC macro 514

## Z

- ZMEAS data collection control 24







File Number: S370/30XX-40  
Program Number: 5748-T14



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SH31-0151-13

