

Transaction Processing Facility



General Macros

Version 4 Release 1

Transaction Processing Facility



General 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 xiii.

Thirteenth Edition (June 2002)

This is a major revision of, and obsoletes, SH31-0152-11 and all associated technical newsletters.

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

IBM welcomes your comments. Address your comments to:

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

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

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

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

Contents

Figures	ix
Tables	xi
Notices	xiii
Trademarks	xiii
About This Book	xv
Who Should Read This Book	xv
How This Book Is Organized	xv
Conventions Used in the TPF Library	xvi
How to Read the Syntax Diagrams	xvii
Related Information	xx
IBM Transaction Processing Facility (TPF) 4.1 Books	xx
IBM Systems Network Architecture (SNA) Books	xxi
Online Information	xxi
How to Send Your Comments	xxi
General Macros Introduction	1
Register Conventions	1
Executive Macros	1
Declarative Macros	2
Macro Usage Conventions	3
BEGIN and FINIS Macros	3
Enter-Back Macros	4
Main Storage Allocation Macros	5
File Storage Allocation Macros	6
File Type Macros	8
File Macros	9
Find Macros	9
General Data Set Macros	10
ROUTC Macro	12
Send Macros	12
Create Macros	12
Event Facility—EVNTC, POSTC, EVNWC, EVINC, and EVNQC	14
Resource Sharing Facility—ENQC and DEQC	15
Alternate Resource Sharing Facility—CORHC and CORUC	15
Wait, Exit, Delay Macros	15
Tape Macros	16
General Tape Operations	17
Unit Record Macros	21
Control Program (CP) Detected Errors	22
Multiple Database Function (MDBF) Macros	24
General Macros	25
ADDLDC—Add Doubleword Unsigned	26
ADDRC—Add Doubleword Unsigned	27
ALASC—Get an Auto Storage Block	28
ALPHA—Alphabetic Scan	31
AMSSC—Error Recording	33
ATTAC—Attach a Detached Working Storage Block	34
BACKC—Return to Previous Program Record	36
BEGIN—Begin Assembler Program	37

BPKDC—TPF Input Message Tokenization Support	40
BPPSC—Static Positional List Build	56
CALOC—Reserve and Initialize Storage	57
CC—Set Condition Code in Current PSW	59
CCIDC—Convert CPUID to Processor Ordinal Number	60
CENVC—Access Environment Variables	61
CIFRC—Cipher Program Interface	64
CINFC—Control Program Interface	66
CM0ND—Build Standardized Scan Tables	70
CM0PR—Scan Input Message for Keywords	72
CNOSC—TPF/APPC Change Number of Sessions Macro	76
CNOSC CHANGE	84
CNOSC DISPLAY	87
CNOSC INITIALIZE	90
CNOSC RESET	95
CONKC—Configuration Constants	99
CORHC—Define and Hold Resource	101
CORUC—Unhold Resource	103
CREDC—Create a Deferred Entry	105
CREEC—Create a New ECB with Attached Core Blocks	107
CREMC—Create a New ECB for Immediate Entry	110
CRESC—Create New Synchronous ECBs	112
CRET—Create a Time-Initiated Entry	116
CREXC—Create a Low Priority Deferred Entry	120
CRUSA—Test and Release Data Level	122
CSONC—Convert System Ordinal Number	125
CYCPC—PSMS Utility Interface Macro	127
DATAS—Combination Data	130
DATEC—Compute Date Stamp	132
DBSAC—Attach TPF Application Requester Database Support Structure	134
DBSDC—Detach TPF Application Requester Database Support Structure	136
DDATA—Terminal Data Field Display	138
DECBC—Manage Data Event Control Blocks	145
DEFRC—Defer Processing of Current Entry	149
DEQC—Dequeue from Resource	150
DETAC—Detach an ECB Working Storage Block	152
DLAYC—Delay Processing	154
DPANL—Terminal Panel Display	155
DPROC—Program Record Determination	163
EDITA—Edit and Move Data	164
ENQC—Define and Enqueue Resource	169
ENTDC—Enter a Program and Drop Previous Programs	171
ENTNC—Enter a Program with No Return Expected	173
ENTRC—Enter Program with Expected Return	175
EVINC—Increment Count for Event	177
EVNQC—Interrogate Event Status	179
EVNTC—Define Internal Event	181
EVNWC—Wait for Event Completion	187
EXITC—Processing of an Entry Is Complete	190
FA4X4C—Convert a File Address	191
FAC8C—Calculate an 8-Byte File Address	193
FILEC—File a Record	195
FILKW—File Keyword	198
FILNC—File a Record with No Release	200
FILSC—File a Single Record	203
FILUC—File and Unhold a Record	205

FINDC—Find a File Record	208
FINHC—Find and Hold a File Record	210
FINIS—Finish Program Assembly	213
FINSCL—Find a Single File Record	214
FINWC—Find a File Record and Wait	216
FIWHC—Find and Hold a File Record, and Wait	219
FLIPC—Interchange the Status of Two Data Levels	222
FREEC—Release Storage Blocks	224
FSTIC—File Status Table Information	226
GCFCL—Get Core Block and Large File Address.	228
GCFSC—Get Small Core Block and File Address	230
GDSNC—Get General Data Set Entry	232
GDSRC—Get General Data Set Record	235
GENLC—Generate a Data List	237
GENMSG—Generate Message Table for WTOPC	242
GETCC—Get a Working Storage Block.	244
GETFC—Get File Pool Address and Storage Block	248
GETLC—Get Large File Storage Address	252
GETPC—Get Program and Lock in Storage	254
GETSC—Get Small File Storage Address	258
GFSCC—Initiate GFS Control	260
GLMOD—Change Global Protect Key	262
GLOBZ—Define Global Fields	264
GLOUC—Request Keypoint Filing.	266
HASHC—Calculate Record Slot Number	268
IGTCCB—Get a Conversation Control Block Entry.	269
INQRC—Convert Resource Application Interface	271
IPSVE—Create a PSV Table Entry	275
IPSVT—Define the Start and End of the PSV Table	277
ITPNT—Transaction Program Name Table Macro	278
ITRPC—Send Simple Network Management Protocol User Trap	283
KARMA—SLC Link Alarm	285
KEYRC—Restore Protection Key	287
LEVTA—Level Test	289
LISTC—Dump Facility LIST Generator	291
LODIC—Check System Load and Mark ECB.	294
LONGC—Set Entry Maximum Existence Time	301
MALOC—Reserve a Storage Block	303
PNAMC—Move Program Name into Specified Field	305
POSTC—Mark Event Completion	306
PRLNC—Print a Line	309
RAISA—General File Get File Address	311
RALOC—Change Reserved Storage Block Size	314
RCATC—Find an RCAT Entry	316
RCHKA—Record Code Check	318
RCRFC—Release Core Block and File Address.	319
RCUNC—Release Core Block and Unhold File Record	321
RDCDC—Read a Card.	323
REHKA—Rehook Core Block	325
RELCC—Release a Core Storage Block	327
RELFC—Return File Pool Address	329
RELPC—Release Program From Storage Lock	331
RIDCC—RID Conversion	333
RLCHA—Release Chain	336
ROUTC—Route a Message	338
RTCUC—Record Type Conversion Utility	340

SAWNC—Wait for Event Completion, Signal Aware	345
SCANA—Scan Data Field.	349
SELEC—Select a Thread Application Interface	353
SERRC—System Error.	357
SIZBC—Obtain Logical Size	362
SNAPC—Snapshot Dump.	363
SONIC—Obtain Symbolic File Address Information	366
SUBDLC—Subtract Doubleword Unsigned	370
SUBDRC—Subtract Doubleword Unsigned	371
SYNCC—Synchronize Globals	372
SYSEQC—Configuration-Dependent System Equates	377
SYSRA—Application System Error	378
SYSTC—Test Sysgen Options	381
TASNC—Assign General Tape	385
TBSPC—Backspace General Tape and Wait	387
TCLSC—Close a General Tape.	390
TDSPC—Display Tape Status	392
TIMEC—Compute Time Stamp	395
TMCNA—Time Conversion	397
TMSEC—Epilog for ISO-C Functions Calling TPF Macro Services	399
TMSPC—Prolog for ISO-C Functions Calling TPF Macro Services	401
TOPNC—Open a General Tape	406
TOURC—Write a Real-Time Tape Record and Release Core Block	409
TOUTC—Write a Real-Time Tape Record	411
TPPCC—TPF/APPC Conversation Verb Macro	414
TPPCC ACTIVATE_ON_CONFIRMATION	427
TPPCC ACTIVATE_ON_RECEIPT	431
TPPCC ALLOCATE.	435
TPPCC CONFIRM	440
TPPCC CONFIRMED	443
TPPCC DEALLOCATE	445
TPPCC FLUSH	449
TPPCC GET_ATTRIBUTES.	451
TPPCC GET_TYPE	454
TPPCC POST_ON_RECEIPT	456
TPPCC PREPARE_TO_RECEIVE	459
TPPCC RECEIVE	462
TPPCC REQUEST_TO_SEND	468
TPPCC SEND_DATA	470
TPPCC SEND_ERROR	474
TPPCC TEST.	478
TPPCC WAIT	481
TPRDC—Read a General Tape Record.	484
TREWC—Rewind General Tape and Wait	486
TR SVC—Reserve General Tape	488
TSYNC—Synchronize Tape	490
TWRTC—Write a General Tape Record.	492
TXBGC—Begin a Global Transaction	494
TXCMC—Commit a Global Transaction.	496
TXRBC—Roll Back a Global Transaction	498
TXRSC—Resume a Global Transaction.	500
TXSPC—Suspend a Global Transaction	502
TYCVA—Time Conversion	504
UATBC—MDBF User Attribute Reference Request	506
UNFRC—Unhold File Record	510
UNHKA—Unhook Core Block	512

URCTC—Unit Record Control	514
USURC—Assign Unit Record Devices	517
UXCMC—User Exit Control	519
UXMAC—Define Multiple User Exit Interface	523
VALBC—Validate Storage Block Address	528
VCHKC—Check or Wait for Request to Be Completed	531
VCLSC—Close a Data Set	533
VENDC—End a Request	535
VGENC—Generate an Access Method Control Block or Request Parameter List	537
VGETC—Get a Record.	544
VIPAC—Move a VIPA to Another Processor	546
VOPNC—Open a Data Set	548
VPNTC—Point for Access.	550
VSHOC—Display a Control Block	552
WAITC—Suspend Processing for ECB I/O Completion	554
WGTAC—Locate Terminal Entry	556
WTOPC—Edit and Send System Message	559
Index	583

Figures

1.	File Address Reference Word	17
2.	Core Block Reference Word.	18

Tables

1. MDBF Macros	24
2. TPF/APPC Change Number of Session Verbs and Valid Keywords	77
3. CNOSC Primary Return Codes	80
4. CNOS Secondary Return Codes	81
5. Correlation of Primary Return Codes to CNOSC Verbs	82
6. Correlation of Secondary Return Codes to CNOSC Verbs	83
7. ITRPC Error Return	283
8. Priority Class Table (Shutdown Levels)	299
9. Output List.	334
10. TPF/APPC Conversation Verbs and Valid Keywords	415
11. TPPCC Primary Return Codes	421
12. TPPCC Secondary Return Codes	422
13. Miscellaneous TPPCC Symbolic Values	425
14. Return Codes for Conversation in SEND State	475
15. Return Codes for Conversation in RECEIVE State	476
16. Return Codes for Conversation in CONFIRM State.	476
17. Return Codes for TEST=POSTED	479
18. Return Codes for TEST=RTSRCVD	480

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:

APPN
ECKD
IBM
VisualAge.

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

About This Book

The two macro books are the primary references for assembler language macro usage under the TPF system. They contain the detailed specifications for all TPF macros. Not included are system installation macros, system macros, data macros, structured programming macros, and macro groups that are related to specific functions or user tasks.

TPF General Macros contains descriptions of macros used by TPF applications programmers. Macros used by systems programmers are described in *TPF System Macros*. These macros comprise those that provide system services (such as performing alphabetic scans, using tape drives, or handling input and output), those that control application programming processing (such as entry creation), and controlling events for resource sharing (such as posting and waiting, queuing and dequeuing). The TPF Advanced Program-to-Program Communications (TPF/APPC) macros are also described in this book.

The application programming interface for TPF specifies that the interfaces to the general macros remain stable. Note in contrast that the interfaces to system macros are not guaranteed.

Macros restricted for use by the TPF system, those macros requiring authorization, or for macros for use in the control program only are documented in *TPF System 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 application and system programmers.

How This Book Is Organized

The first section of *TPF General Macros* contains a general introduction to macro usage and the usage conventions for the more common macro types. There is also information about how to read syntax diagrams. This is followed by the detail specifications for each macro. The detail specifications are ordered alphabetically by macro name.

General use macros are available for use by all programs. They provide the interfaces required to use various services of the TPF system.

TPF development pursues all feasible options to prevent making changes in these macros that would require any user changes in application code.

TPF Advanced Program-to-Program Communications (TPF/APPC) macros available for general use are provided in a separate section to preserve continuity in the book.

The following macros are described outside this book:

- **System macros** for TPF system use are in *TPF System Macros*.
- **Structured programming macros** for application programs are in *TPPDF 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 the book.

Conventions Used in the TPF Library

The TPF library uses the following conventions:

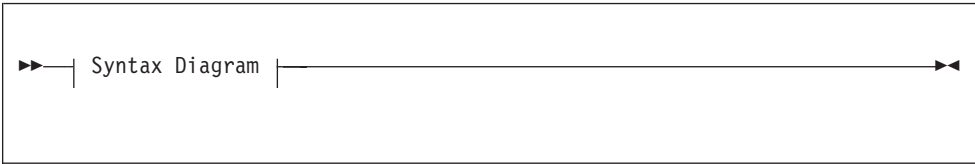
Conventions	Examples of Usage
<i>italic</i>	Used for important words and phrases. For example: <i>A database is a collection of data.</i> Used to represent variable information. For example: Enter ZFRST STATUS MODULE <i>mod</i> , where <i>mod</i> is the module for which you want status.
bold	Used to represent text that you type. For example: Enter ZNALS HELP to obtain help information for the ZNALS command. Used to represent variable information in C language. For example: level
monospaced	Used for messages and information that displays on a screen. For example: PROCESSING COMPLETED Used for C language functions. For example: maskc Used for examples. For example: maskc(MASKC_ENABLE, MASKC_IO);
<i>bold italic</i>	Used for emphasis. For example: You <i>must</i> type this command exactly as shown.
<u>Bold underscore</u>	Used to indicate the default in a list of options. For example: Keyword=OPTION1 <u>DEFAULT</u>
Vertical bar	Used to separate options in a list. (Also referred to as the OR symbol.) For example: Keyword=Option1 Option2 Note: Sometimes the vertical bar is used as a <i>pipe</i> (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=option

Conventions	Examples of Usage
Scale	<p>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.</p> <p> ...+....1....+....2....+....3....+....4....+....5....+....6....+....7...</p> <p>LOADER IMAGE CLEAR</p> <p>Notes:</p> <ol style="list-style-type: none">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.

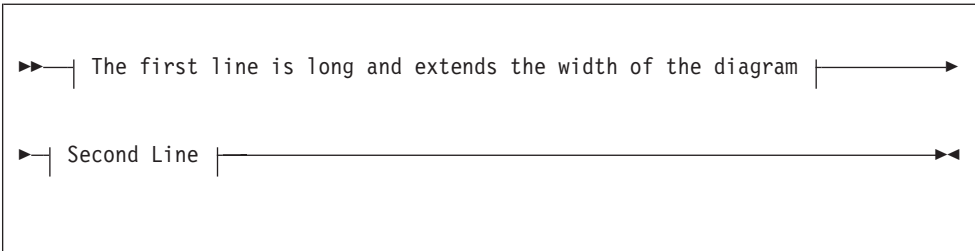
How to Read the Syntax Diagrams

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

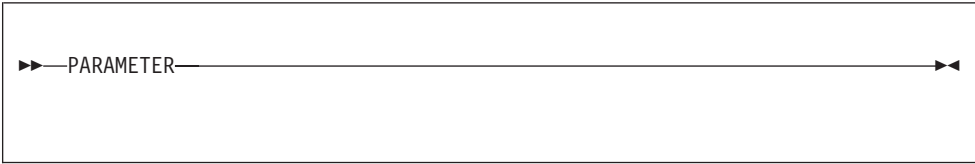
- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads and ends on the right with 2 arrowheads facing each other.



- 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.

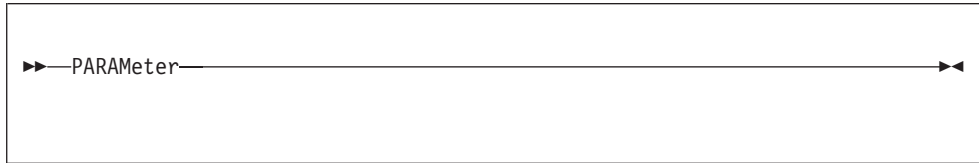


- A word in all uppercase is a parameter that you must spell **exactly** as shown.

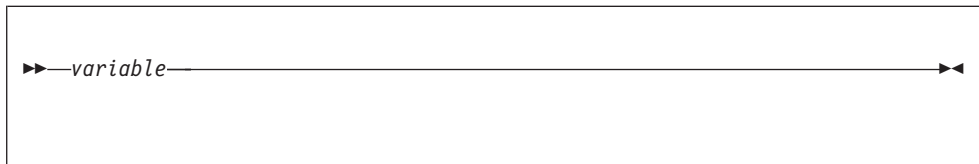


- If you can abbreviate a parameter, the optional part of the parameter is shown in lowercase. (You must type the text that is shown in uppercase. You can type none, one, or more of the letters that are shown in lowercase.)

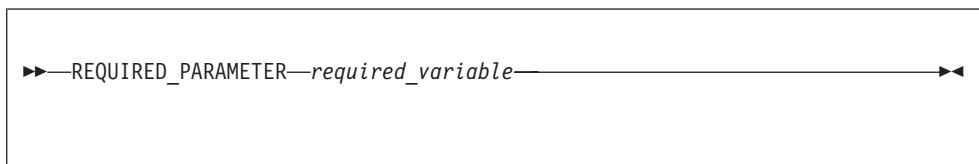
Note: Some TPF commands are case-sensitive and contain parameters that must be entered exactly as shown. This information is noted in the description of the appropriate commands.



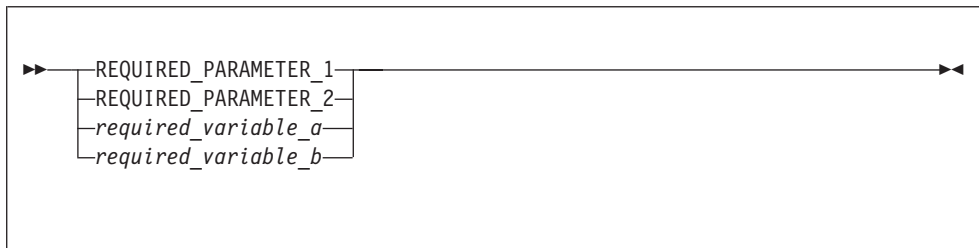
- A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.



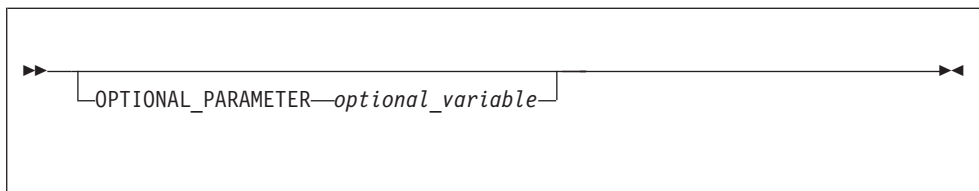
- Required parameters and variables are shown on the main path line. You must code required parameters and variables.



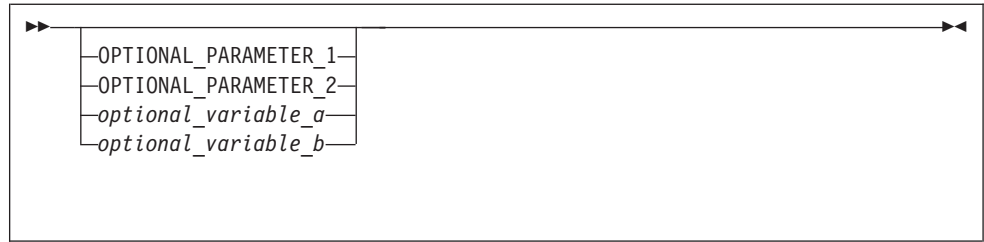
- If there is more than one mutually exclusive required parameter or variable to choose from, they are stacked vertically.



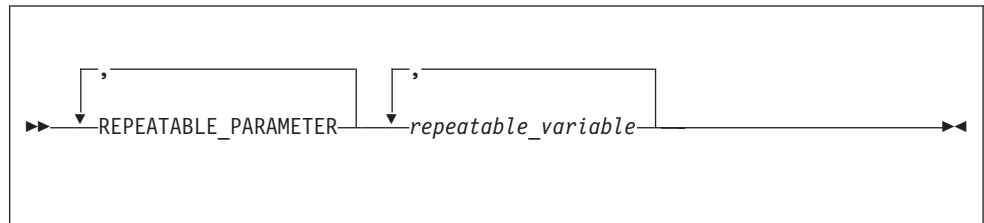
- Optional parameters and variables are shown below the main path line. You can choose not to code optional parameters and variables.



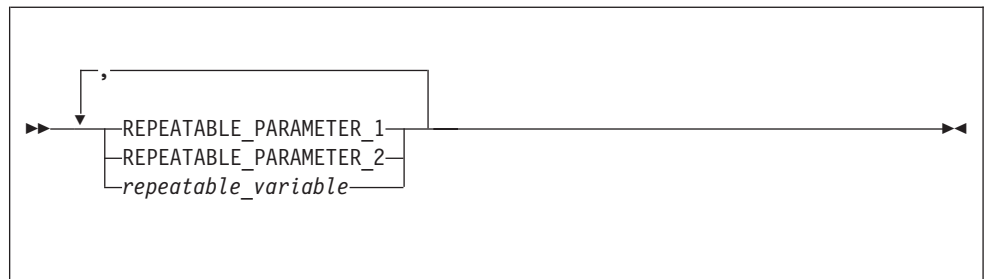
- If there is more than one mutually exclusive optional parameter or variable to choose from, they are stacked vertically below the main path line.



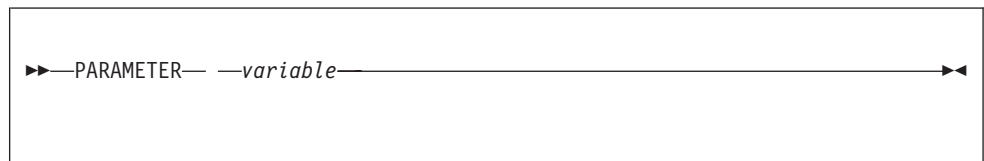
- An arrow returning to the left above a parameter or variable on the main path line means that the parameter or variable can be repeated. The comma (,) means that each parameter or variable must be separated from the next parameter or variable by a comma.



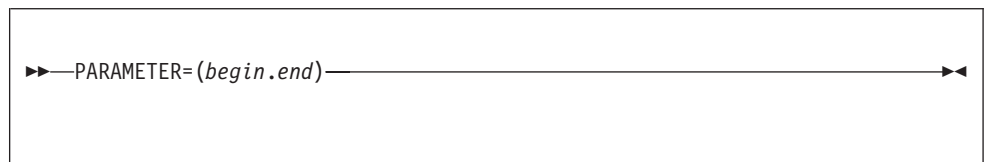
- An arrow returning to the left above a group of parameters or variables means that more than one can be selected, or a single one can be repeated.



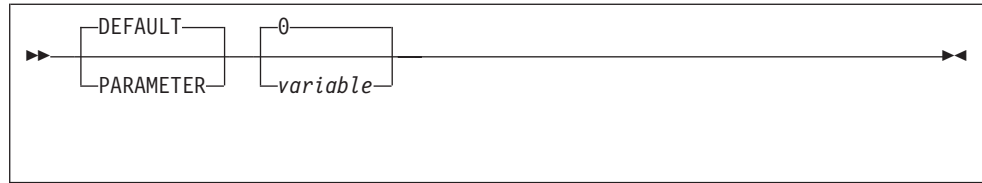
- If a diagram shows a blank space, you must code the blank space as part of the syntax. In the following example, you must code **PARAMETER** variable.



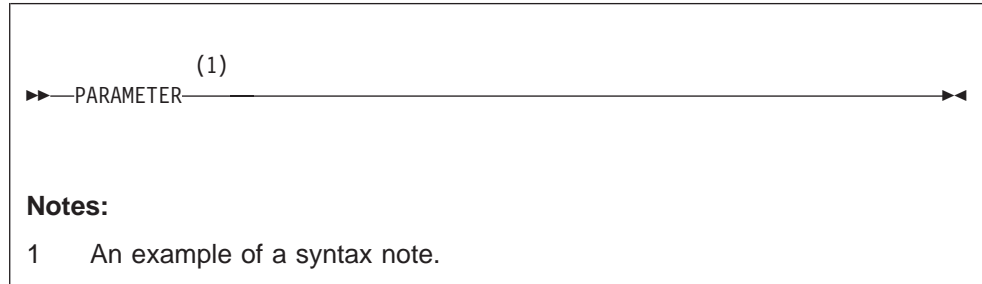
- If a diagram shows a character that is not alphanumeric (such as commas, parentheses, periods, and equal signs), you must code the character as part of the syntax. In the following example, you must code **PARAMETER=(begin.end)**.



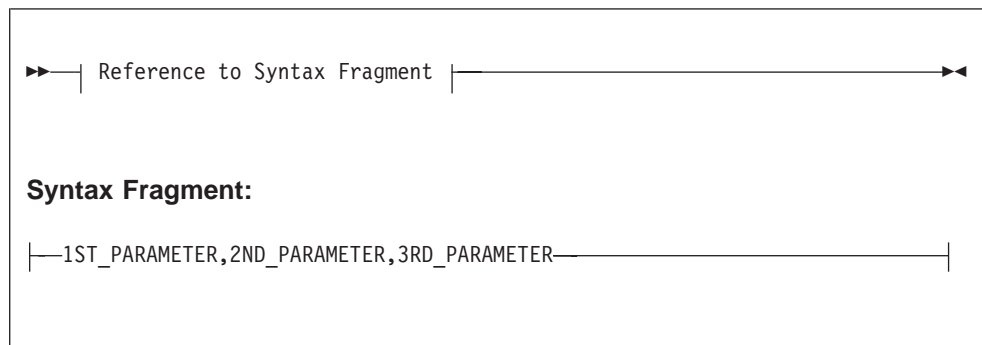
- Default parameters and values are shown above the main path line. The TPF system uses the default if you omit the parameter or value entirely.



- 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 ACF/SNA Network Generation*, SH31-0131
- TPF Application Programming*, SH31-0132
- TPF C/C++ Language Support User's Guide*, SH31-0121
- TPF Database Reference*, SH31-0143
- TPF Data Communications Services Reference*, SH31-0145
- TPF Migration Guide: Program Update Tapes*, GH31-0187
- TPF Operations*, SH31-0162
- TPF Program Development Support Reference*, SH31-0164
- TPF Programming Standards*, SH31-0165
- TPPDF and TPF Structured Programming Macros*, SH31-0183

- *TPF System Generation*, SH31-0171
- *TPF System Installation Support Reference*, SH31-0149
- *TPF System Macros*, SH31-0151.
- *TPF Transmission Control Protocol/Internet Protocol*, SH31-0120

IBM Systems Network Architecture (SNA) Books

- *IBM Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *IBM Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808
- *IBM Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084.

Online Information

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

How to Send Your Comments

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

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

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

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA
- If you prefer to send your comments by FAX, use this number:
 - United States and Canada: 1 + 845 + 432 + 9788
 - Other countries: (international code) + 845 + 432 +9788

General Macros Introduction

In the TPF environment, *macro* is a generic term that covers several types of user services. Macro definitions are used to generate the requests for the TPF system online control program services. Such a request is normally expanded into an SVC instruction and necessary parameters, which are ultimately interpreted by the TPF system service routines invoked by the online macro decoder. For example, a FINDC macro request processes an application (Entry) to have the system obtain (find) a record from the online DASD files.

Register Conventions

Registers usage is an important part of programming with macros. In the TPF system certain registers have specific uses:

Register	Usage
R8	E-type program base register.
R9	Entry Control Block (ECB) base register.
R10 - R13	Control Program base registers

These registers should not be used by application programs. Also various registers are guaranteed to remain the same across macro calls unless noted otherwise. These are R0 – R9.

In general, there are two types of macros:

Macro Type	Description
Executive	These macros generate either code or data that is incorporated into the program being assembled. Generally, an executable instruction is generated.
Declarative	These macros produce information used by the assembly process while generating code.

TPF system programs utilize a large set of macro instructions to generate linkage(s) or to simply generate *inline* code. Many of these macros are restricted to system programs because the macros are subject to change in future releases and represent an unprotected interface. A macro with an unprotected interface is called a *restricted use macro*. Because some TPF system programs run in the application execution environment, some of the restricted use macros also generate SVC linkages.

Executive Macros

The executive macros are divided into two general groups: control program macros and application macros. All executive macro names consist of 5 alpha characters. The last letter of the macro name distinguishes between control program macros (C) and application macros (A). A few application macro names violate this general rule.

Control program macros are used to provide a linkage to the system service routines. Generally, this implies the generation of a supervisor call (SVC) instruction and parameters that specify the desired service. The macro decoder routine of the

control program, in the case of an SVC, selects the appropriate control program routine to perform the service. In some instances, the linkage is done with branch and link instructions to improve the performance of executing frequently used functions.

Application macros can generate a sequence of machine instructions. A control program macro generally generates only enough code to link to the desired service routine. An application macro can generate data or indirectly generate linkage code through the use of a control program macro.

If an application macro definition uses a control program macro, a linkage will, of course, be generated for the system service routines. The distinction is:

- The application macro definition uses a control program macro as an inner macro.
- The SVC machine instruction is never directly used by an application macro definition.

Some of the application macros are specifically designed for the offline programs. Another subset is specifically designed for building structured programs.

The format of the executive (control program and application program) macro instruction used in source code is as follows:

Label	An optional symbolic label that can be assigned to the macro instruction
Name	The symbolic macro instruction name
Symbols	Macro parameters, depending on the specific macro used.

There are 3 major macro instruction operand (parameter) formats.

Positional	The macro instruction operands must be written in a fixed order. We assume that they precede any keyword parameters unless otherwise noted.
Keyword	The macro instruction operands can be written in any order and are keyed to a specific character set (for example, TEST, PARAMETER).
Mixed Mode	The macro instruction operands are a combination of both positional and keyword operands. That is, certain operands (positional) must be written in a fixed order; other operands (keyword) can be specified in any order.

Declarative Macros

The declarative type macros can be divided into three general groups: Data macros, Equate macros, and SIP macros.

Data macros are used to declare the symbolic names used by the source code of programs to refer to fields in some data record. DSECTs are generated by data macros. Any source code that includes the DSECT and uses names declared by the DSECT will have the proper offset values assigned to object code on completion of the assembly process.

Equate macros are used to handle the declaration and assignment of values to system names during the assembly process (offset values for data level fields in the

entry control block (ECB), which are names such as D0 and D1, in many control program macros). An equate macro is used to name a collection of assembly EQU instructions. In this way, operational programs can have access to large blocks of system values using a single program statement. The format of an equate macro instruction is simply the name of the equate macro (for example, SYSEQ). Examples of types of data used in equates are the record types in the fixed file, offset values for the data levels in the ECB that are referred to symbolically as D0–DF, maximum lines of output in a given type of display. By using equates, the absolute values can be changed in the macro without having to modify the operational programs (the programs need only be reassembled).

SIP macros are used by the System Initialization Program for processing parameters unique to an installation. They are not used in any other programs.

Macro Usage Conventions

This section provides additional programming considerations and general use information on selected groups of commonly used macros. This information complements the detail specifications in the remainder of the publication.

In the years when TPF was confined below the 16M line, addresses were 24 bits long. Since words and registers were 32 bits long, sometimes the extra 8 bits were used improperly for non-addressing purposes. With the advance of addressing past the 16M line the extra bits can be easily misinterpreted, giving incorrect results. In particular, some macros can return incorrect results if addresses passed to them have their high order bit (X'80') set. The technique of loading a zero address; such as,

```
LA Rx,0(Rx)
```

before a macro call clears out registers in the way required.

A note about severity codes for macros. Some conditions warrant attention from users. Usually these conditions are errors, however, sometimes they are informational comments. Error severities range from 4 (for the least severe) to 12 (for the most severe). Informational comments are listed with a severity of *.

BEGIN and FINIS Macros

The BEGIN and FINIS macro instructions must be used as the first and last source statement, respectively, in any assembler language ECB-controlled program. These macros generate header information in assembled object code. The header is used by the system allocator and system loader to load the programs to the TPF system online modules from the loader general file. This header represents a system record to different initialization procedures.

- The BEGIN macro assigns values to the following program header items:
 - Program name
 - Program version
 - Program reference (*external* symbols resolved by the TPF online loader)
 - Program record size
 - Program base register
 - Program CSECT name.

The BEGIN macro also calls a data macro, EB0EB, which declares the symbolic names used by ECB-controlled programs to reference fields within the ECB. SYSEQ and some other equate macros are also called by the BEGIN macro.

- The FINIS macro:
 - Ensures the presence of a BEGIN macro.
 - Verifies the CSECT name.
 - Calls 2 equate macros, REGEQ or CPSEQ, which define general register names and generally used symbolic names.
 - Assigns the program size into object CPDE during the assembly process. This is removed by the time the program is loaded to the online files.

Note: Many of the executive macros depend on various fields in the ECB. Therefore, it is very useful for the programmer to have detailed knowledge of this control block prior to reading this publication.

Enter-Back Macros

The Enter-Back macros provide the facility for ECB-controlled (also referred to as E-type) programs to transfer control to other E-type programs. The control program performs the processing necessary to transfer control, including any I/O operations.

The E-type programs can reside on direct access storage devices (DASDs), in VFA buffers, or in main storage.

If the program to which control is to be transferred is resident on an external storage device, a Read operation is performed to retrieve the program. While the Read operation is taking place, the control program can give control to another ECB if one is ready for processing (implied Wait).

A program can request transfer of control to another program by issuing a kind of Enter-Back macro. The following requests can be made:

- Enter Program with Expected Return (ENTRC macro). The requesting program remains attached to the ECB.
- Return to Previous Program Record (BACKC macro). The returning program is released from the ECB.
- Enter a Program with No Return Expected (ENTNC macro). The requesting program is released from the ECB.
- Enter a program and Drop Previous Programs (ENTDC macro). The requesting program and all other programs attached to the ECB are released from the ECB.

The transfer of the information required by the control program to locate and invoke a requested program is accomplished using the Enter-Back macro expansions.

In general, the macro expansions contain:

- Linkage instructions to pass control to the macro handler routine in the control program
- Program pointer to either the program's PAT entry or the program name.

The resolution of the macro expansions is performed in two stages. The first stage takes place when the issuing program is assembled. At that time, space is reserved in the program for the macro expansion by V-type constants (V-CONS). The V-CONS not only reserve space, but the label that is used identifies the type of macro and the name of the requested program (see the ENTRC, ENTNC, ENTDC, and BACKC macro descriptions).

The second stage occurs when the issuing program is selected for loading to the Loader General File. At this time the Linkage Editor program is invoked to resolve the V-CONS contained in the program to be loaded. The Link Editor program uses the contents of the program allocation table (PAT), which is a directory of all of the system's E-type programs, to determine the type of residency of both the requesting and requested program. The address of the requested program is also obtained from the PAT. Using the data from the PAT, the Link Editor program fills in the Enter-Back macro expansions.

There are several classifications of the E-type programs. Programs that are allocated to reside on DASD devices are referred to as *file-resident* programs. E-type programs can also be allocated to reside in main storage. Programs allocated to main storage are classified as *core resident* programs.

Main Storage Allocation Macros

Main storage can be obtained and ultimately released in any of the following ways.

- Running a Get Working Storage Block (GETCC) macro causes the control program to assign a block of main storage of the specified size. The block is associated and held by a data level in the entry control block. Blocks of storage obtained in this manner can be released through execution of a Release a Core Storage Block (RELCC) macro.
- Running a Find a File Record (FINDC) macro will also cause the control program to assign a block of main storage. In this case, the storage is associated with the entry just prior to starting the Read operation. Blocks of storage obtained in this manner can be released through execution of a Release a Core Storage Block (RELCC) macro or through the execution of a File a Record (FILEC) macro. This returns the block of storage to the working storage pool automatically following completion of the Write operation.
- Running a Read a General Tape Record (TPRDC) macro will also cause the control program to assign a block of main storage. In this case, the storage is associated with the entry after the Read operation has been completed. Blocks of storage obtained in this manner can be released by running a Release a Core Storage Block (RELCC) macro or running the Write a General Tape macro (TWRTC) and Write a Realtime Tape Record and Release Core Block macro (TOURC), which return the block of storage to the working storage pool automatically when the Write operation is completed.

When an operational program executes a GETCC macro, the control program will assign one main storage block of the specified storage size to the entry control block. Reference to the main storage block is placed in the core block reference word for the specified level.

The RELCC macro allows the operational programs to release main storage blocks after they are no longer needed. The control program returns the main storage block (referenced in the core block reference word) to the working storage pool when an RELCC is executed. Only one main storage block can be returned at a time. Main storage blocks should be released as soon as possible by the user program.

In general, an operational program should use a GETCC macro to obtain a main storage block in which a data record or message is to be created. Following creation of a message or data record, a File or Send type macro is used to transmit the data to auxiliary storage or over a communication line.

Use care when using these macros. To allow for efficient utilization of main storage, return blocks as soon as they are no longer required by the user program. The control program releases all blocks held by an entry when the entry exits.

File Storage Allocation Macros

A file pool storage record address can be obtained by executing a GETFC macro for a 4KB, large, or small record respectively. The file pool record address is placed in the file address reference word (FARW) of the data level specified in the macro statement.

For GETFC, the record ID implies an index into the RIAT (DCRIT) table from which record attributes are obtained.

The basic file pool types optionally available in a system include:

- Small, long-term pool (SLTx)
- Small, short-term pool (SSTx)
- Small, long-term duplicate pool (SDPx)
- Large, long-term pool (LLTx)
- Large, short-term pool (LSTx)
- Large, long-term duplicated pool (LDPx)
- 4KB, long-term pool (4LTx)
- 4KB, short-term pool (4ST)
- 4KB, long-term duplicated pool (4DPx)
- 4KB, long-term duplicated FARF6 pool (4D6).

Where x is A, B, C, or D and indicates that the pool types are distinguished by the TPF system devices (that is, DEVA, DEVB, DEVC, and DEVD).

Pools 1, 2, 4, 5, 7, and 8 will contain single copy records in a partially duplicated file system and duplicate records in a completely duplicated file system. Pools 3, 6, and 9 are provided for systems with a partially duplicated file system and a requirement for duplicate file pool storage records.

All file pool record addresses must be returned by executing a RELFC macro. The file address reference word (data level) containing the address that is being returned is specified as a parameter of the RELFC macro. Short-term pool addresses are, in effect, immediately returned to the relevant pool for reuse. Consequently, short-term pool records can be used to store different records during a short interval of time. Long-term pool ordinals and records are stored in a return block that is written to a Real-time tape for offline processing. The returned long-term addresses can be kept out of the online system for several days.

It is suggested that the short-term pools be used for storing records that are maintained for short intervals of time (for example, the time required to complete a reservation transaction with a customer). The short-term pools are designed for quick turnover records. Long-term file pool records can be maintained during an interval of time as dictated by operational procedures.

The basic file pools previously mentioned can be split across more than one external storage device type. Each split is called a *pool section*. The selection of a pool section is normally based on the record ID in the RIAT for the GETFC macro. Ratio dispensing/pool fallback can modify normal pool selection as described later.

The following sample macro statement will be processed as described:

```
CVEDXX  GETFC D2,,CW
```

1. Assembler Program

```

CVEDXX    DC    VL3 (CWL#Z6)
           DC    AL1 (D2)

```

2. Link Editor Program

Record ID CW is assigned the following characteristics:

- It is a large record.
- It is a single-copy record.
- It must be maintained for several days.
- It must be located on DEVA (DASD).

Based on the these assumptions, the VL3 will be resolved as follows:

Byte	Value	
0	0A	SVC Code
1	2	LLT pool
2	04	DEVA (DASD)

The GETFC macro expands as follows:

Byte	Bit	Contents
0		SVC
1		GETFC INTERRUPTION CODE
2	0	= 0 RECORD ID IN MACRO EXPANSION = 1 RECORD ID IN SPECIFIED REGISTER
	1	= 0 PRIME RECORD ATTRIBUTE = 1 OVERFLOW RECORD ATTRIBUTE
	2	= 0 DON'T GET CORE BLOCK = 1 GET CORE BLOCK
	3	= 0 SYSTEM ERROR IF INVALID REC ID = 1 RETURN IF INVALID RECORD ID
	4-7	SPECIFIED REGISTER (IF BIT 1 ON)
3		DATA LEVEL INDEX
4,5		HEXADECIMAL REC ID (IF BIT 1 OFF)

GETFC has an implied Wait. When a Get File Storage request cannot be immediately serviced, further processing of the entry requesting the file address is postponed. A file pool directory is reordered when an active directory reaches a predefined critical level of available addresses. The implied Wait is applied to a Get File Storage macro when the appropriate file pool directory is depleted of available addresses. This allows processing of other entries in the system.

All file storage addresses should be returned as soon as possible with a RELFC macro. A file storage address must be released only once. If a short-term double release occurs, a system error dump is taken. Control is then returned to the operational program that executed the RELFC macro.

Ratio dispensing is used to dispense addresses for a specific basic pool type using several sections of the pool. As each pool section is selected, its ratio number (that is, the number of addresses) is used to determine how many addresses to dispense from that section before selecting another pool section.

File pool fallback allows alternate pool sections with like records (that is, small or large, duplicate or single copy) to be used to satisfy GETFC requests to depleted pool sections. Short-term pool sections can fall back to short-term or long-term pool sections. Long-term pool sections can fall back only to other long-term pool sections with like records.

Ratio dispensing and pool fallback are available for pool sections. The schedules used to control this function are initialized based on optional SYCON parameters or commands. See *TPF Database Reference* for more information about ratio dispensing and pool fallback.

File Type Macros

The file macros allow the user programs to Read (Find), Write (File), and Hold a file record. They consist of the following:

FDCTC	File Data Chain Transfer
FILEC	File A Record
FILSC	File A Single Record
FILUC	File A Record And Unhold
FILNC	File A Record And No Release
FINDC	Find A File Record
FINHC	Find A File Record And Hold
FINSC	Find A Single File Record
FINWC	Find A File Record And Wait
FIWHC	Find A File Record Hold And Wait
FLSPC	File A Special Record
FNSPC	Find A Special Record
UNFRC	Unhold File Record.

Operational programs address file records by supplying a reference record identification (ID) and a pseudo file address in a file address reference word (FARW). Each record ID field is contained in 4 bytes of main storage in the ECB. Each record ID field in a data event control block (DECB) is contained in 4 bytes of main storage (like an ECB) and the file address field is 8 bytes (for a total of 12 bytes of main storage).

The format of the record ID field is as follows:



The record type identifies the general type of the record requested (such as inventory and passenger name index). The record code check (RCC) is a unique number generated by the operational programs and is used by the control program to verify the integrity of records. The following describes the various requirements and checks made by the control program.

File Macros

1. If the RCC = 0, no verification is made by the control program. If the RCC = nonzero, the control program compares the RCC fields in the ECB or DECB to that of the record to be filed. If the comparison fails, a system error is generated and the ECB is exited.
2. The record type must always be supplied and can never be zero. The control program compares the record type field in the ECB or DECB to that of the record to be filed. If the record type is zero or the comparison fails, a system error is generated and the ECB is exited.

Find Macros

1. If the RCC = 0, no verification is made by the control program. If the RCC = nonzero, the control program compares the RCC field in the ECB or DECB and the record that was read. If the comparison fails, the appropriate system error bits (CE1SUD, CE1SUG in the ECB, or IDECSUD in the DECB) are set. No system error is generated and return is made to the user program following WAITC processing.
2. If the Record Type = 0, no verification is made by the control program. If the Record Type = nonzero, the control program compares the record type field in the ECB or DECB and the record that was read. If the comparison fails, the appropriate system error bits (CE1SUD, CE1SUG in the ECB, or IDECSUD in the DECB) are set. No system error is generated and return is made to the user program following WAITC processing.

Note: Each of the 16 data levels in an ECB has a detailed error indicator (CE1SUD). In addition, there is a gross data level indicator (CE1SUG), which represents the cumulative error indicator for all the data levels. With TPF DECB support, each DECB has a detailed error indicator (IDECSUD). The existing gross data level indicator (CE1SUG) accounts for any errors that occurred on a DECB-related I/O operation. Therefore, it is not necessary to have a separate gross indicator for TPF DECB support. When an error occurs for a DECB, bit IN1DUD in the ECB indicator byte (CE1IN1) is turned on.

See *TPF Database Reference* for more information about the formats of the TPF file addresses. The format of the file address is determined by the type of FIND/FILE macro issued and the value of the GDS keyword specified on the FIND or FILE macros. For example, a FNSPC or a FLSPC macro supports only a hardware address format of module number (MM), cylinder number (CC), head number (HH), and record number (R), whereas a FINDC macro can be used to access general files, general data sets and the online database. The file address format is determined by the GDS keyword on the FINDC macro. In TPF3, the code to handle a FIND or FILE for a GDS or GF checked two places to differentiate between a normal FARF3 SON or a GDS or GF file address format. In TPF4, the GDS parameter must be set to YES to distinguish between FARF3 SON addresses and general file or general data set addresses. Earlier versions of TPF did not require this. Not doing so, even in FARF3 systems, can cause addressing failures.

Duplicate Operations

The control program examines the file address and, following the execution of a Write (FILEC) macro, both the primary and duplicate record will be updated automatically if both exist. Likewise, following the execution of a Read (FINDC) macro, the control program will retrieve the alternate record if the primary record cannot be successfully read. Only if neither the duplicated nor the primary record is found will an error be posted.

Occasionally, it is necessary for a program to selectively read or write only the primary or duplicate record. A set of unique macros (FINSC, FILSC) is provided for this purpose, and unlike other file macros:

1. They should be used only to access records from the online database.
2. Only the specified record (primary or duplicate) is updated following execution of *Write single*.
3. Only the specified record (primary or duplicate) is retrieved following execution of *Read single*.

Holding a Record

The *record hold* feature is a method of reserving data for exclusive use of a program during the updating of that record. When a Hold macro is used, the Record Hold Table is checked to determine if the record is being held for a previous request. If the record is not being held, the file address of the record is placed in the Record Hold Table and the request is serviced. If a previous request is holding the required record, the current request is not serviced until the record is unheld by the previous entry. All record hold requests are serviced on a first-come, first-served basis.

When executing a Find a File macro (FINDC), it is possible to access a record that is currently being held and to update it. The Record Hold Table is checked only when servicing a Find and Hold (FINHC, FIWHC) macro.

Note: Therefore, it is imperative that all programs updating records observe the convention of using a Find and Hold macro to retrieve the record to be updated. The TPF control program does **not** enforce this convention.

An entry should hold only one record at a time. This is necessary to avoid a condition where two entries are each waiting for a record that the other is holding. A record can be unheld by executing an Unhold macro (UNFRC, FILUC). A held record must be unheld before processing of the entry is completed.

It is important to recognize that running any write File macros (except FILNC) will cause the ECB data level or DECB containing the main storage block to return to the available storage pool after completing the write operation. In addition, the data block is detached from the entry at macro processing time. Accordingly, following the running of any File macro (except FILNC), the data level or DECB is available on return from the control program even though the write operation can be completed.

General Data Set Macros

The General Data Set Name (GDSNC) and General Data Set Record (GDSRC) macros are used to access general data set records. GDSNC is used to open and close the data set. GDSRC is used to access specific records in the data set. The Find and File type macros are used to read from and write to the data set.

You must open the data set by passing the name, volume sequence number, and relative record number (RRN) of the data set with the GDSNC macro. The volume sequence number and RRN can be zero. GDSNC returns the indexes necessary for the system to access the data set that the user specified with the macro (in fields CE1FMx, CE1FCx, CE1FHx, and CE1FRx of the ECB data level or fields IDECFM, IDECFH, and IDECFR of the DECB). The file address of the RRN in the data set is returned in CCHR format in the file address extension word (FAXW) (field CE1FXx for the ECB data level or field IDECFX0 for the DECB). If the RRN was zero, the address of the first record of the data set is returned.

To access specific records in the data set the user must pass the RRN of the desired record with the GDSRC macro. The RRN can be zero. GDSRC returns the file address of the desired record in the FAXW (CE1FXx for the ECB data level or IDECFX0 for the DECB) in CCHR format. If the RRN specified was zero, the address of the first record of the data set is returned.

The Find or File macros (FDCTC, FILEC, FILNC, FILUC, FLSPC, FINDC, FINHC, FINWC, FIWHC, FNSPC, UNFRC) are used to read data from, or write data to, the general data set record. You can set up the FARW (CE1FAx for the ECB data level or IDECFRW for the DECB) with the record ID and RCC. The file address was already set up by the GDSNC and GDSRC macros.

After managing the data with the Find and File macros, you must close the general data set with the GDSNC macro. The parameters passed are the same as those passed with GDSNC open. The FARW (fields CE1FMx, CE1FCX, CE1FHx, CE1FRx for an ECB data level or fields IDECFM, IDECFC, IDECFH, and IDECFR for a DECB) is cleared on return from GDSNC close.

An example of the use of the General Data Set macros follows:

```
*****
* "OPEN" THE GENERAL DATA SET.                                     *
*****
      XC   CE1FM8(1),CE1FM8      VOLUME SEQUENCE NUMBER
      MVC  EBW000(16),=CL16'TPF.DATA.SET.A'  DATA SET NAME
      MVC  EBW016(4),=F'0'      RELATIVE RECORD NUMBER
      LA   R14,EBW000           ADDRESS OF DATA SET NAME
      GDSNC D8,0,RCT=A,SIZE=L,WORK=YES  OPEN DATA SET
      LTR  R14,R14              CHECK RETURN CODE
      BNZ  GDSNERR              BRANCH IF ERROR
*****
* READ, UPDATE, AND WRITE THE FIFTH RECORD OF THE GENERAL DATA SET. *
* WHEN WORK=YES THE USER MUST PASS THE RELATIVE RECORD NUMBER IN THE *
* FOUR BYTES IMMEDIATELY FOLLOWING THE DATA SET NAME.                *
*****
      LA   R14,EBW000           ADDRESS OF DATA SET NAME
      MVC  EBW016(4),=F'5'      RELATIVE RECORD NUMBER = 5
      GDSRC D8,SIZE=L,WORK=YES  GET ADDRESS OF 5TH RELATIVE RECORD
      LTR  R14,R14              CHECK RETURN CODE
      BNZ  GDSRERR              BRANCH IF ERROR
      MVC  CE1FA8(2),=CL2'GS'   RECORD ID
      MVC  CE1FA8+2(1),=XL1'00' RECORD CODE CHECK
      FINDC D8,GDS=YES          READ DATA SET RECORD
      WAITC FINDERR             BRANCH IF ERROR
      .
      .                          UPDATE THE DATA SET RECORD
      .
      FILEC D8,GDS=YES          WRITE DATA SET RECORD
      WAITC FILEERR             BRANCH IF ERROR
*****
* READ, UPDATE, AND WRITE THE THIRD RECORD OF THE SAME DATA SET.    *
* WHEN WORK=NO THE USER MUST PASS THE RELATIVE RECORD NUMBER IN THE *
* FARW EXTENSION.                                                      *
*****
      LA   R14,EBW000           ADDRESS OF DATA SET NAME
      MVC  CE1FX8(4),=F'3'      RELATIVE RECORD NUMBER = 3
      GDSRC D8,SIZE=L,WORK=NO    GET ADDRESS OF 3RD RELATIVE RECORD
      LTR  R14,R14              CHECK RETURN CODE
      BNZ  GDSRERR              BRANCH IF ERROR
      MVC  CE1FA8(2),=CL2'GS'   RECORD ID
      MVC  CE1FA8+2(1),=XL1'00' RECORD CODE CHECK
      FINHC D8,GDS=YES          READ DATA SET RECORD
      WAITC FINDERR             BRANCH IF ERROR
      .
```

ROUTC Macro

The application program identifies the recipient by giving the symbolic terminal or application program name in a routing control parameter list, RCPL (RC0PL). When the ROUTC macro is issued, the system determines which CPU in the network hosts the receiving terminal or application program and then forwards the message to that CPU. Upon receipt of the message, the host CPU transmits it to the terminal or activates the application program.

Send Macros

Send macros should be avoided whenever possible. They should be issued only by system programs. The Send macros are device-specific while Route macros require no knowledge of the receiving device. So, user-supplied programs and, where possible, system programs should use the ROUTC macro. It is also recommended that one control output program per user application be given ROUTC responsibility to minimize impact if future upgrading occurs.

Create Macros

CREDC	Create a deferred entry
CREMC	Create an immediate entry
CRETC	Create time initiated entry
CREXC	Create a low-priority deferred Entry
CREEC	Create an immediate entry with a core block
CRESC	Create a synchronous entry

12 TPF V4R1 General Macros

The CREDC macro should be used to create an entry and defer its processing to a nonpeak period. The operational program has the ability to initialize the created ECB work area with as many as 96 bytes of parameter information. These parameters can be set up at execution time because the address and byte count of the parameter area is specified in scratch registers R14 and R15. The control program buffers the parameters in an available main storage block and adds this block to the deferred processing list (serviced only in a low activity period). When the deferred list is serviced, Operational Program Zero gets an ECB, moves the parameters into the work area, and releases the interim main storage block. At this time an Enter with No Return (ENTNC) macro is executed, transferring control to the program specified by the CREDC macro. It is important to understand that the repeated use of the CREDC macro can cause a depletion of main storage. When there is insufficient storage to buffer the parameters, the control program forces the ECB issuing the CREDC into a WAITC. The macro will be executed when the necessary main storage becomes available.

The CREMC macro is similar to the CREDC macro except that the interim main storage block buffering the parameters is added to the postinterrupt list. Because excessive use of the CREMC can lock out input messages to the system, this macro should be used carefully.

Operational programs that are above 1052 state can create time-initiated entries by executing the CRETC macro. The control program remembers this request and transfers control to the specified program at the correct time. An ECB is assigned to the created entry when the new program is started. A control block can be passed to the created entry using the LEVEL parameter. The CRETC macro differs from the CREMC and CREDC macros in that the CRETC parameter area is limited to one word, which appears as the first word of the work area in the created ECB. The time interval (3 bytes), in minutes or seconds, that indicates the time relative to the present time, must be specified by the operational program. Both the time interval and the action word are dynamic parameters in that they are set up at execution time in scratch registers R14 and R15. The information required by the control program to retrieve the specified program is specified in the macro expansion.

The CREXC macro is similar to the CREDC macro, but it is more specialized in that it requires more main storage blocks to be available before it will execute. It should be used by programs that create many entries. The use of this macro by doing many *creates* ensures that there is no degradation of the real-time system by a created shortage of main storage blocks.

The CREEC macro is similar to CREDC and CREMC usage and processing with the additional capability of passing a core block to level zero of the created entry control block. The operational program specifies whether a deferred or immediate entry is to be created and follows the processing paths of CREDC and CREMC, respectively.

Use the CRESC macro to create a synchronous entry. The entry is created in the same subsystem and subsystem user as the issuing program. The issuing program can specify a timeout value to indicate how long the program will wait for the created entry to end. When exiting, the created entry will notify the issuing program by posting completion of an event for which the program is waiting. The issuing program can create as many as 50 entries by coding multiple CRESC macros that specify WAIT=NO, followed by a CRESC WAIT=YES call. The program can pass as much as 4 KB of data to each created entry and can specify on which I-stream to start the entry.

Note: Limit the use of this macro to prevent storage depletion.

Event Facility—EVNTC, POSTC, EVNWC, EVINC, and EVNQC

This facility allows ECB-controlled programs to define in the control program a subsystem-unique named event, which can be waited on and posted by all ECBs that know its name and data base ID (DBI). This name can be any 8-character, EBCDIC value. Events coded by IBM (*) start with either "SYS" or "I".

The current design allows for three types of trigger mechanisms:

1. A count of POSTs that have to be done against the named EVENT before it is satisfied.
2. A 16-bit mask that triggers the event when the mask is reduced to zero by having the event's bits ANDed off by the POST mechanism.
3. A by-product of the count mechanism that allows the ECB to pass a core block to a second ECB.

The application program creates a named event by issuing the EVNTC macro. The control program assigns an entry in the internal event table and formats it with the information provided by the user. The ECB determines whether a named internal event exists by issuing the EVNQC macro. If the event is not found, processing branches to a path indicated for events not outstanding. When the ECB is ready to wait, it issues the EVNWC macro. To post the named event, an ECB calls the POSTC post event macro. When it is called, the control program searches the internal event table for an event entry with the same name. If no match is found, a nonzero condition code is set and return is made to the application. If a match is found, the type of POSTC is compared against the event type, and if it is not equal, a system error results and the ECB is exited. If the type is a count, the count in the event is decremented by one for the POSTC. If the type is a mask, the mask supplied by the POSTC is ANDed against the mask in the event.

For both count and mask events, another supplied mask is ORed into the return field for the event. If the POSTC macro also specified an error, the error byte is set into the event block and the event is considered triggered even though the mask or count is nonzero.

If the type is core block (CB), the core block reference word (CBRW) from the specified level is copied into the event block and considered to be triggered. A return is made to the posting task with a zero return code.

If the event was triggered and an EVNWC macro has been issued for the event, the event block is placed on the ready list with a postinterrupt address to cause entry into the event completion routine. The event completion routine moves the mask fields back into the event area specified by the EVNWC macro, and it moves the error indicator into the SUD field for the level or event area if level was not specified. The I/O counter is decremented and, if zero, the application program is activated.

On EXITC, any events created by the exiting ECB are destroyed unless another ECB is already waiting for the event, in which case ownership is transferred to the waiting ECB.

As part of the EVNTC macro, a TIMEOUT value can be specified. If the system is above 1052 state, the event will be timed from the time of the first EVNWC until either the event is triggered or the specified time period passes. If the time period passes first, the event is posted complete with error.

After an event has been defined by the EVNTC macro, the EVINC macro can be used to increment dynamically the count for a count-type event. Note, however, that an EVNWC macro cannot have already been issued for the event.

Resource Sharing Facility—ENQC and DEQC

This facility allows multiple ECBs to share a named resource and to serialize access to the resource. To use this facility, a common name has to be agreed on by the programs using it. This name can be any 8-character value. For IBM-written programs, if the name is in EBCDIC, the first 3 characters should be SYS.

To update a shared resource, an ECB issues the ENQC macro with the name of the resource as the parameter. If it is not already in the control program's table, the name is added and the ECB is marked as holding a resource. If the name is in the control program's table, the ECB is queued on the resource entry and a Wait macro is initiated until the resource is freed by the holding ECB. When the holding ECB completes its processing on the shared resource, it issues a DEQC macro for the resource. This frees the resource and allows the control program to transfer ownership to the waiting ECB. If there are no waiting ECBs, the resource entry is cleared.

Alternate Resource Sharing Facility—CORHC and CORUC

This facility is similar to ENQC and DEQC usage and processing. The differences are:

- The resource name is only 4 characters.
- The resource name is always systemwide.
- There is no time-out capability.

These macros are supplied to provide compatibility for existing applications.

Wait, Exit, Delay Macros

User programs use a Find a File macro (FINDC) to retrieve data stored on the files. However, the data record can be referred to by the user program upon return from the FINDC macro. A WAITC macro is used to guarantee that the input operation has been completed. Only upon return from the WAITC macro can the user program assume that the data record is now in storage and available. The delay for completion of an I/O operation is often substantial when compared to CPU processing speed. Accordingly, following a WAITC macro call, and during the delay pending I/O completion, control can be transferred to another program (or to the current program) for processing of another entry.

The WAITC macro is normally used to ensure completion of an input operation. On output operations (transmit a message or update a data record) the data record is detached from the entry block during calls to the File, Send, or Route macros. Use of the WAITC macro does not ensure completion and the user program cannot determine when output operations are completed. A special mode of operation is provided for the TOUTC macro where a WAITC macro must be called to ensure completion of the output operations. When message processing has been completed, an EXITC macro is called. This program returns all main storage blocks held by this entry to the available storage pools (including the ECB itself), therefore, in that way ending the life of the entry in the system.

Operational programs can defer processing of a low priority task until the activity in the system is sufficiently low to warrant calling it. The Defer (DEFRC) macro is provided for this purpose. The delay prior to service of such entries can be

substantial. Accordingly, no records can be held by an entry calling a DEFRC macro and no time-dependent (that is, terminal) response should follow this macro. If the entry is holding considerable working storage, calling of this macro can adversely affect system throughput. The Delay (DLAYC) macro differs from the Defer macro only in that such entries are added to the CPU input list. This list has a higher priority than the CPU defer list.

Note: Defer and Delay macros are normally called by file maintenance or exception routines, and are not recommended for general use.

Tape Macros

Real-time and General Tape macros provide the operational program with single file, multivolume tape input and output. In order to fulfill different system requirements, two distinct I/O capabilities are provided. The real-time tapes are intended for logging system changes, system performance, and other dynamic maintenance information in the real-time environment. The general tapes provide the tape writing, reading, and searching capabilities required by utility programs.

All tapes need a symbolic 3-character name. The first 2 characters must be alphabetic, and the third must be alphabetic or numeric. Additionally, the first 2 characters for a general tape name cannot be RT.

Realtime Tape Operations

The real-time tapes are *Write only* tapes that are available to all entries in the system. All hardware tape functions such as writing labels, checking labels, and detecting end-of-volume conditions are handled by the control program. The operational program is relieved of these responsibilities and can assume that the real-time tapes are always available. Records are written on these tapes in the order in which the control program receives TOUTC and TOURC macros. Each record is identified by the subsystem and time stamped. However, because each tape is also available to other entries in the system, an operational program cannot depend on creating consecutive records.

TOUTC Macro

The TOUTC macro writes a record contained in main storage, on the specified real-time tape. This record must be at least 1 byte and no more than 32K bytes length, and must not be altered until the writing operation is complete. The execution of a WAITC macro ensures the completion of the I/O transfer. The record can be in a permanent storage resident area, a working storage block, or the fixed work area in the ECB. To prevent a condition where the tape output record is modified by another entry prior to the completion of the I/O transfer, it is recommended that the output record be located in a storage area held exclusively by the ECB.

When using a TOUTC macro, an operational programmer must communicate the following information to the control program:

- The symbolic name of the real-time tape to be written on
- The starting location of the record to be written
- The number of bytes to be transferred. This byte count cannot be 0.

These parameters are passed to the control program through the macro expansion and a specified file address reference word (FARW). The macro expansion identifies the request level and the symbolic tape to be used. The starting location of the record and the byte count are stored in the specified request level in the following format:

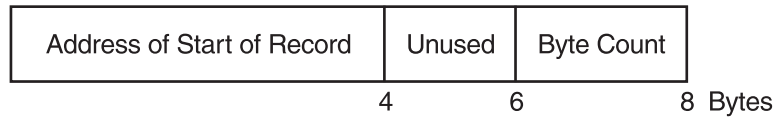


Figure 1. File Address Reference Word

Core Block Reference Word: The core block reference word is unused. Therefore, the operational program can use any request level without releasing the main storage block held.

TOURC Macro

The TOURC macro writes a record from main storage to the specified real-time tape. The core block must be held by the entry control block at the specified core block reference word. This main storage block is detached from the entry control block at macro time and the writing proceeds independently. Consequently, the operational program cannot determine the status of the Write operation and the call of a WAITC macro is meaningless. Moreover, the request level used in the TOURC macro is immediately available for more use after return from the macro.

Tape Pseudonyms

Real-time tape names (like RTA and RTB) can be given aliases, or tape pseudonyms. Tape pseudonym RTM, for example, can be mapped to RTA using an RTMAP definition in CEFZ. Such a mechanism provides a means for mapping many names to just a few tape devices.

Both real-time and general tape names may be used as real-time tape pseudonyms. However, it is not recommended to use general tape names as real-time tape pseudonyms. This is because some tape macros (TDSPC and TSYNC) can be used for both general tapes and real-time tapes. For these macros, an attempt is first made to resolve the specified tape name (which may be a real-time tape pseudonym) to a real-time tape name. If the tape name is successfully resolved then the real-time tape name is used by the macro service routines; otherwise the tape name is treated as a general tape name. Note that tape macros that are used exclusively by real-time tapes (for example, TOURC or TOUTC) or exclusively by general tapes (for example, TOPNC or TPRDC) do not exhibit this behavior. This can lead to confusion if not properly understood.

For example, suppose the general tape name ABC is mapped to the RTM real-time tape. If the operator mounts a ABC tape and an application issues a TSYNC NAME=ABC macro, expecting to synchronize the ABC tape, the RTM tape will be synchronized instead. This is because the real-time tape pseudonym translation is done first.

Only one level of tape name resolution is performed. For example, if the tape pseudonym RTM is mapped to RTB, and the tape pseudonym RTB is mapped to RTA, then TOURC NAME=RTM will write to the RTB tape, not the RTA tape.

General Tape Operations

This section discusses macros for general tape operations.

Open and Close Macros

A number of active general tapes can be on the system at any time. These tapes are separated into distinct sets or groups, each set exclusively associated with a particular active entry. No entry can use another entry's general tapes concurrently. In addition to the tape reading, writing, and searching capabilities provided, the

operational program has the facility to define and adjust the definition of its tape set. The Open a General Tape (TOPNC) and Close a General Tape (TCLSC) macros are used for this purpose. A general tape first becomes available to an operational program through the use of the TOPNC macro. The 3-character symbolic general tape name, specified in the TOPNC macro, is used to identify this tape for all general tape macros. When a TOPNC macro is called, the tape name is added to the tape set for that entry. The control program performs the specified labeling functions and presents the desired tape positioned at the initial data record. The TCLSC macro deactivates the specified tape by deleting its tape name from the entry's tape set. Trailer labeling is performed by the control program when the TCLSC macro addresses an output tape.

The running of utility operational programs and allocation of physical tape units for general tape use are controlled by the system operator who is responsible for mounting the proper tapes and initiating the various job phases in their correct sequence. The operator must know where new input or scratch tapes are needed and when output tapes are complete.

Input/Output Macros

The general tape input/output macros are Read a General Tape Record (TPRDC) and Write a General Tape Record (TWRTC). In addition to the I/O operation, each macro performs the appropriate main storage allocation. During a call of the Read macro, the control program gets a main storage block to read the record into. Reference to this main storage block is placed in the specified main storage block reference word when the input operation is complete. Conversely, after the Write macro is completed, the main storage block containing the record is released to the appropriate storage pool. Accordingly, each request for tape I/O by the general tape macros has an associated request level in the ECB. This request level contains the following information:

File Address Reference Word: Unused

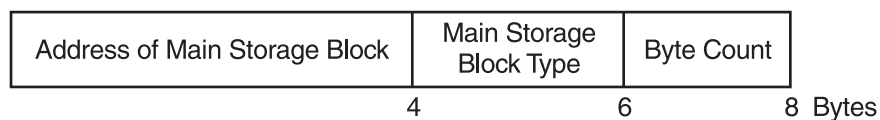


Figure 2. Core Block Reference Word

On return from the TPRDC macro, the status of the Read operation is unknown. Consequently, a call to the WAITC macro ensures completion of the I/O. At Wait Return time, the condition code is used to indicate the success or failure of any completed input operation.

On a TWRTC macro, the writing operation and the main storage block containing the data record are detached from the ECB at macro time. Therefore, the request level used is immediately available for use after the return from a TWRTC macro.

Condition Code Settings: The condition code is used with the branch on condition generated in the WAITC macro expansion. It results in a transfer to an operational error routine when an unusual condition or I/O hardware error is detected. At WAITC return time, the condition code has one of two settings: zero or nonzero.

A setting of zero indicates that all input operations (that is, TPRDC macros) were completed as correct length record reads (CLR). On a CLR, the size of the physical

record on tape is equal to the size of the main storage block requested at macro time. In this case, control passes to the instruction following the WAITC macro expansion.

A nonzero setting indicates that an unusual condition or I/O hardware error has been detected during at least one of the completed TPRDC macros. In this case control is transferred to the operational error routine by the branch on condition generated in the WAITC macro expansion.

For each I/O hardware error or unusual condition, the control program records an identification character in the entry control block by request level (CE1SUD). Consequently, the operational program can determine which I/O requests were successful, which were unsuccessful, and the nature of the error, by examining this field in the ECB.

Wrong Length Records: At WAITC return time on wrong length record TPRDC macros, the associated core block reference word will be holding a main storage block with the requested record. The byte count for this request level contains the number of bytes read. For a short-length record (SLR), this count will be less than the normal byte count of the requested main storage block. For long-length records (LLR), the count will equal the normal byte count of the requested main storage block (the record will be truncated). The condition code is nonzero and the request level indicator has one of two settings: Short-Length Record (10) or Long-Length Record (08).

End-of-File Condition: A TPRDC macro, which results in an end-of-file (EOF) condition, is aborted in the following manner. At Wait return time the core block reference word corresponding to the TPRDC macro is not holding a main storage block. The condition code is nonzero, and the end-of-file bit is on (04) in the associated request level indicator (CE1SUD).

A number of TPRDC macros can be issued to the same symbolic tape before a WAITC macro. If the first macro call results in an end-of-file (EOF), the remaining macros are aborted in the same manner as the first. If the last macro call results in an end-of-file (EOF), only the last will be canceled. At Wait Return time, the condition code is nonzero and the associated request level indicators in the system error word reflect the completion status of each macro. Only the successfully completed macros will be holding main storage blocks containing the requested records.

Invalid Format Condition: when a TPRDC macro results in an invalid format condition, the ECB is terminated with a system error dump. The block in error causing the condition is displayed in the dump. This error comes about when a variable formatted block is either corrupted on the tape or does not correspond to its own record descriptors. It can occur too when a tape with variable records is defined as having fixed records.

Tape Hardware Error Condition: A dump is taken when errors occur due to equipment failure. The dump and any associated messages should be examined to determine the cause of the problem.

Assign and Reserve Macros

For single entry utility jobs (that is, those that process from start to finish within the same entry control block), the previously described macros provide an adequate macro library. However, some jobs with extensive running time are segmented into a sequence of short phases. Each phase is a unique entry control block in the system and is initiated by the computer operator upon completion of the previous

phase. In such a case it is desirable to have one set of tapes associated with the entire life of the job. An example of such a job might be nightly file maintenance, in which a number of capabilities are desirable:

- One input tape—processed in sequence by each phase of the job.
- One output tape—processed in sequence by each phase of the job.

To eliminate operator tape handling between job phases, the operational programmer can use 2 special purpose macros: Reserve General Tape (TRSVC) macro and Assign General Tape (TASNC) macro. These macros are designed to pass an open set of general tapes from an active entry, which is about to exit, to a future entry.

Between these phases the positioning of the tapes remains unchanged. The TRSVC macro reserves the specified tape in the set of this entry or current job phase for use by a future entry or the next phase. When the next phase of the job is started, the new entry makes the tapes available by using a TASNC macro. The following example can clarify the use of the TRSVC and TASNC macros.

Phase I:

Macro	Tape	Comments
TOPNC TOPNC TOPNC	SKC,I SKB,I SKA,0	These tapes have been mounted by the operator; they are rewound and entered in the tape set as active tapes.
Tape I/O Processing by Phase I		
TCLSC	SKB	General tape SKB is removed from set, rewound, and unloaded.
Tape I/O Processing by Phase I		
TRSVC TRSVC EXITC	SKA SCK	General tapes SKA and SKC are reserved for Phase II; their numbering and positioning remain unchanged.

Phase II:

Macro	Tape	Comments
TASNC TASNC	SKA SKC	Tapes SKA and SKC are assigned to the ECBs issuing the TASNC macro. Processing continues on each tape from its position at the time of the last TRSVC macro.
Tape I/O Processing by Phase II		
TCLSC TCLSC EXITC	SKC SKA	Output tapes SKA and SKC are closed: trailer-labeled, rewound, and unloaded.

Operator Control of Tape Operations

The following is a method of structuring general tape programs to allow for independence from the hardware tape configuration. The operational programmer provides the operator with a run sheet outlining the symbolic tape configuration for each job. This run sheet identifies each tape by general name and type (input, output, scratch, and label information). Using the run sheet, tape units are assigned and the necessary tapes are mounted for the job. The operator then communicates the tape units assigned and their corresponding general tape names to the control program.

Each tape is recorded as being in a ready status by the control program. When the utility job is started the operational program opens each tape with a TOPNC and general tape name. The control program expects to find a ready tape for each TOPNC issued by the operational program. If no such tape exists, the control program sends a message to the operator requesting the desired tape. Return from the TOPNC macro is delayed until the tape is made ready. Only the operator and the control program are concerned with the tape units in use. Therefore, the operational program, working with symbolic general tape names, is independent of the system's physical tape configuration.

Tape Labeling and Multivolume Operations

General tapes used on the online system can be labeled or nonlabeled tapes. All general and real-time output tapes that are used on the online system have header and trailer labels. The control program performs all labeling and label checking independent of the operational programs. The format of the tape labels is consistent with IBM tape label standards. Header labels are checked on all tapes and are written on all output tapes when the tapes are mounted on the system. Mounting informs the control program that this tape is available for use by initializing the tape status table. Mounting can be performed by the operator in response to a TOPNC macro or in anticipation of a request to open a specific tape. The control program writes trailer labels during end-of-volume processing and on TCLSC macros for output tapes. In addition, trailer labels are checked on input tapes to distinguish between end-of-volume and logical end-of-file.

The transition from volume to volume for all input and output tapes is handled automatically by the control program. The operational programs are not aware of this transition and are dealing with a continuous symbolically addressed logical file.

Unit Record Macros

The unit record macros are used by programs to obtain a record from, or output a record to, a unit record device. The macros can be divided into the following categories:

Resource Allocation

USURC	This macro assigns devices to the requesting ECB for its life in the system if no other ECB is using the device. Bit 7 of the first byte CE1URM of the field in the ECB is set to 1 to indicate that a unit record device is assigned to this ECB. This macro must be called before any of the unit record I/O macros can be called.
EXITC	This macro contains code to check bit 7 of CE1URM in the ECB to determine whether any unit record device is associated with the ECB issuing the EXITC macro; if on, the device assigned to this ECB is returned to the system for use by other programs.
PRLNC	<p>This macro will print a line of data to the printer specified by the user.</p> <p>A WAITC macro must be called to wait for the completion of the I/O operation. The error branch address in the WAITC macro will be taken if an error is detected by the unit record CSECT during the I/O operation.</p>
RDCDC	<p>This macro causes the next card to be read into the specified storage block from the specified card reader.</p> <p>To ensure completion of the read operation, a WAITC macro must be called before attempting to use the expected record.</p>

Unit Record Control Macro

URCTC This macro is mainly used to perform unit record error recovery functions and some system utility functions such as UCS buffer load and FCB load. The application program should not have to use this macro.

System Error Detected by Operational Programs (OP)

The SERRC macro is used by operational programs to signal the detection of errors found during processing. Normally, calling this macro causes an appropriately identified message to be transmitted to the CRAS terminal, and areas relevant to the active entry are dumped. For some application programs, the SYSRA macro is used to call system error. The application programmer can select from a variety of the following options when using this macro:

- A system error number and prefix must be specified to uniquely identify a processing error. This number will appear both on the CRAS error message and on the associated dump, if any. The default prefix is U.
- A user-created message to be appended to the basic CRAS error notification message can be specified. This message will also appear in the dump listing.
- By means of a user-generated *from-to* list, the user can designate areas of storage to be dumped in addition to those normally included in a dump.
- The user can direct the SERRC processing routines to either return to the next sequential instruction or exit the active entry after dumping. If the exit option is chosen, all attached records will be detached and open tapes will be closed.

Control Program (CP) Detected Errors

During any control program macro calls, a variety of exceptional conditions can be detected. These conditions are described in the paragraphs that follow. The options available to the application programmer, as previously described, are also available to the system programmer with respect to the usage of the SERRC macro. The systems programmer has the additional option of tailoring dumps of system storage using the dump override table.

Macro Parameter Errors

These errors consist of programming errors that cannot be tolerated in an operational system and are therefore considered irrecoverable for this entry. The control program issues a system error macro with an exit option, which results in a main storage dump, a message to the system console, and the elimination of this entry from the system. (Entries from a terminal operator will be sent to program CPSA, which sends a response to the terminal and then exits.)

I/O Hardware Errors

These errors are caused by hardware malfunctions and are initially processed by control program error routines. If these corrective processes are unsuccessful and the error persists, the control program calls a system error macro, which results in a main storage dump and a message to the system console. Following the WAITC, a transfer is made to the user-specified operational error routine, where corrective action can be started. Because a dump has already been taken by the control program for I/O hardware errors, the operational program should not duplicate this operation by calling another system error.

I/O-Associated Unusual Conditions

These include end-of-file or wrong length record on tape macros, and ID or RCC check failures on Find and File macros. In these cases, the only action taken by the control program is to identify the condition in the entry control block and to transfer to the WAITC-specified error routine in the operational program. At this time, the

error routine can assess the severity of the unusual condition and take corrective action if possible. The System Error (SERRC) macro can be used to take a dump and send a message to the system console.

Operational programs regain control only on I/O hardware errors or I/O-associated unusual conditions. These two classes differ in one important respect: the control program takes a dump and sends a message only on I/O hardware errors. Consequently, the operational program error routine should use only the System Error macro on I/O-associated unusual conditions. To aid the operational programs in handling both hardware errors and unusual conditions, the program identifies the condition in the ECB (CE1SUD) before transferring control to the WAITC-specified error routine. There are 2 exceptional condition identification fields in the ECB. They consist of a series of detail indicators (16 bytes, one for each data level and one for the program level) and a summary or gross indicator (1 byte). All indicators (detail and summary) have the same format:

Bit
X'80' - I/O Hardware Error (main storage dump taken)
X'40' - ID Check Failure (IDF) (on read)
X'20' - RCC Check Failure (RCCF) (on read)
X'10' - Short length Record (SLR)
X'08' - Long Length Record (LLR)
X'04' - End of File (EOF)
X'02' - Invalid File Address
X'01' - reserved

Bits 4 and 5 will be set by the tape routines to indicate an End-of-Volume (EOV) condition to the user of the TDCTC macro. This combination will only be set for input tapes. Output tapes will set only the EOF indication (Bit 5).

Bits 0 and 4 are set together to indicate Mod Down (main storage dump has been taken).

A bit equal to 1 indicates the presence of the condition; zero indicates the absence of this condition. Consequently, a zero byte indicates no hardware errors or unusual conditions have occurred.

In cases of IDF, RCCF, SLR, and LLR (bits 1, 2, 3, 4), a main storage block is held on the data level. The user's error routine should check these settings in CE1SUD of the ECB.

Both gross (CE1SUG) and detail (CE1SUD) indicators are set by the control program at WAITC return time. Each detail indicator is associated with a particular request level. When an unusual condition occurs, the detail indicator corresponding to the data level for the macro is set with the appropriate bit configuration (such as EOF and RCCF). In addition, the gross byte is updated to indicate that an unusual condition of a particular type has been detected. A WAITC macro can be issued for multiple I/O requests; accordingly, a number of unusual conditions can be present at WAITC return time. The gross byte is a summary and therefore indicates only that conditions of a given type have occurred. The detail indicators define the status of each I/O macro completed as a result of the WAITC macro. Those macros that have completed without unusual conditions or I/O hardware errors have detail indicators of zero. All detail bytes associated with reference levels that are not pertinent to this WAITC macro are set to zero.

Note: ID or RCC failures on a Write are considered to be software errors resulting in the entry being sent to Exit, whereas if they result from a Read operation, they are considered to be an unusual condition resulting in transfer to the user's error routine.

The user does not need to clear the summary or detail indicators because they are cleared by the control program on finding another WAITC.

Multiple Database Function (MDBF) Macros

The macros required to support a system generated with the multiple database function (MDBF) provide the facility to access resources in a subsystem (SS) other than the one generated for the user. This cross subsystem access can either be directed from an SS or subsystem user (SSU) to the BSS or from the BSS to an SS or SSU. The resources accessed can be one of the following:

- Program base
- Database
 - Fixed file record SS or SSU
 - Pool file record SS or SSU
 - Global data SS or SSU
 - Main storage resident SS data.

Table 1. MDBF Macros

Macro Name	Restriction	Description
CEBIC	System	Change SS or SSU Data Resource ID
CROSC	System	Cross SS Access Service
LEBIC	System	Load (and index) ECB Resource ID
UATBC	General	Request for SSU Attribute Table Reference.
Note: The system macros are described in <i>TPF System Macros</i> and the general ones are described in this publication.		

An example of the use of these macros can be found in the CEBIC macro description in *TPF System Macros*.

General Macros

This chapter contains an alphabetic listing of the TPF general macros. General macros are used by TPF application programs that need system services. No special authorization is required by the macros, although some parameters may be restricted, as noted. The description of each macro includes the following information:

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

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

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

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

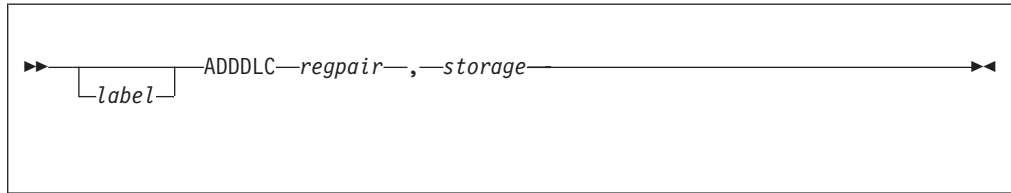
Examples: Provides one or more examples that show you how to code the macro.

See TPF System Macros for information about the TPF system macros.

ADDLDC–Add Doubleword Unsigned

Use this general macro to add two unsigned doublewords.

Format



label

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

regpair

is the even register of an even-odd pair of registers that contain one of the doubleword values to be added, and will contain the result of the addition.

storage

is the label of a storage location that contains a doubleword value that will be added to the doubleword in *regpair*.

Entry Requirements

None.

Return Conditions

The result of the operation will be in *regpair*. The condition code will be set as if an AL assembler instruction was used.

Programming Considerations

None.

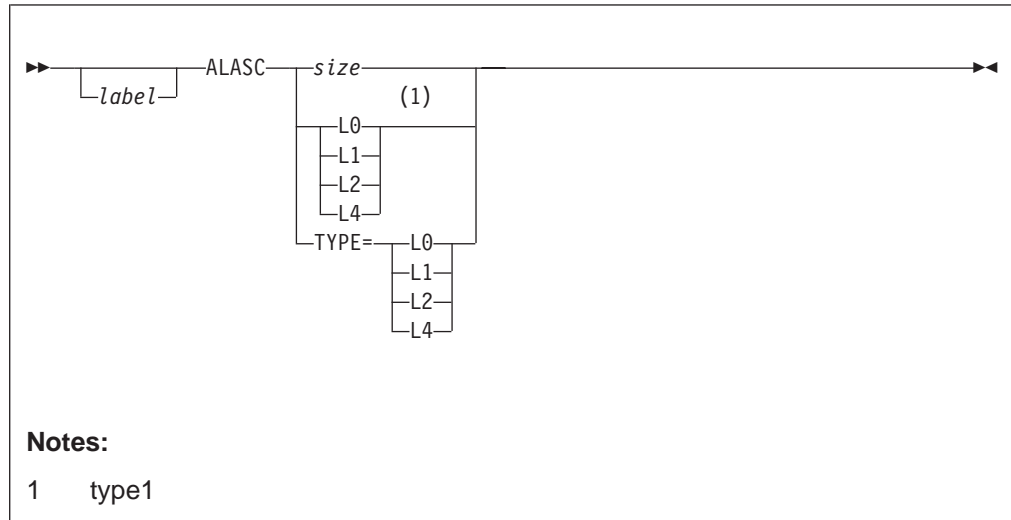
Examples

None.

ALASC–Get an Auto Storage Block

This general macro assigns a storage block from the specified working storage list to the ECB level by placing its address in the auto-storage block reference word (CE1AUT). Only one auto-storage block can be attached to an ECB on each program nesting level.

Format



label

A symbolic name can be assigned to the macro statement.

Either the positional or the keyword parameter may be used.

size

A storage size may be specified symbolically as parameter one. The size specified as the number of bytes, is rounded up to a full double word value (maximum of 4079).

type1

A storage list type may be specified symbolically as parameter one. The ALASC macro will convert this symbol to a size value. Valid list references are:

L0 128 byte block - user size = 120

L1 381 byte block - user size = 376

L2 1055 byte block - user size = 1048

L4 4095 byte block - user size = 4079

TYPE

The argument for this keyword parameter is identical to that required for the *type1* positional parameter.

Entry Requirements

- R9 must contain the address of the ECB being processed.
- The ECB auto storage block reference word must not be holding a storage block for this entry at the current program nesting level in the entry block.

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 except for R7.
- The assigned storage block consists of a header area and a user area, and is initialized as follows:
 - Any earlier auto storage block is chained to this block by placing the contents of CE1AUT in header field CU1ACH.
 - The address of the current ECB program nesting level is placed in header field CU1PNL.
 - The caller's R7 is placed in the first word of user area CU1USR.
 - The remainder of the block is initialized with a unique value determined by the ALASC service routine.
- The address of the assigned block is placed in the auto storage block reference word CE1AUT.
- R7 contains the address of user area CU1USR in the assigned block.

Programming Considerations

- This macro can run from any I-stream.
- Storage blocks held by an operational program will be released when the nesting level drops below the level at which the block was obtained. That is, BACKC, ENTDC, ENTNC, or EXITC will adjust the auto storage levels accordingly.
When using these macros, when returning to the program that calls the ALASC macro, general register 7 addresses the user area in the assigned block of the current program level. The contents of register 7 at the time of the ENTxC/BACKC macro are saved in the first fullword of the user area (CU1USR) in the assigned block.
- Only one auto storage block can be assigned for each program nesting level used by the entry. If a block is already held, a system error dump is taken and the entry is exited.
- Auto storage block labels (CU1ACH,..PNL,..USR) are defined in DSECT CZ4CP.
- ALASC generates a label of the form xxxxyy where xxxx is the name of the calling segment and yy is the version of the calling segment. This label is required by SABRETALK and labels like this must not be coded by users to avoid duplicate label errors.
- ALASC cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES).

Examples

The following example shows the use of ALASC:

- Prior to ALASC call-

```
ECB
    CE1AUT- 0022A008      (current auto-storage block)
    CE1PNL- 00207A68      (current nesting level)
Register R7 - y.....y
Auto Storage Block (at 0022A008)
    CU1ACH- 00000000      (no prior block)
    CU1PNL- 002052D0      (nesting level at attach time)
    CU1USR- x.....x      (caller's R7 at attach time)
    CU1USR+4- 8080.....  (initialization value)
```

- Macro call-

ALASC

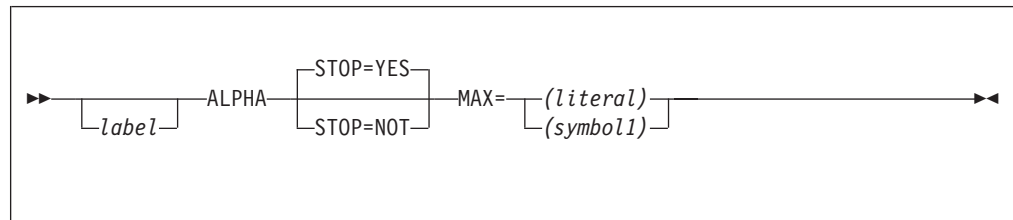
```
      XYZ      ALASC 200      Get a 200 byte auto storage block
• After ALASC call-
  ECB
      CE1AUT- 00235008      (current auto-storage block)
      CE1CPNL- 00207A68      (current nesting level)
Register R7 - 00235010      (CU1USR field)
Auto Storage Block (at 00235008)
      CU1ACH- 0022A008      (prior block address)
      CU1PNL- 00207A68      (nesting level at attach time)
      CU1USR- y.....y      (caller's R7 at attach time)
      CU1USR+4- A0A0.....      (initialization value)
```

ALPHA–Alphabetic Scan

This general macro scans a specified field from left to right to detect the presence of either an alphabetic character (A–Z) or a nonalphabetic character, stopping when the requested character set is found. The Translate and Test (TRT) instruction is used, limiting the size of the field scanned to a maximum of 256 bytes.

After a successful scan for an alphabetic character, the alpha group number, used for passenger names in airline reservation systems is given in R2. (An alpha group is a set of alphabetic characters. The entire alphabet is divided into a series of mutually exclusive alpha groups.)

Format



label

A symbolic name can be assigned to the macro statement.

STOP

This parameter has two possible values:

YES

This causes a scan to stop on the first alphabetic character detected. This is the default.

NOT

This causes a scan to stop on the first nonalphabetic character detected.

MAX

This parameter has either a numeric or symbolic value.

(literal)

A decimal number equal to or less than 256, which defines the number of bytes to be scanned from the address in R1.

(symbol1)

The symbolic address of a general register containing the number of bytes minus 1 (-1) to be scanned from the address in R1. The contents must be less than 256.

Entry Requirements

- R9 must contain the address of the ECB being processed.
- R1 must contain the address of the first byte of the field to be scanned.
- R2 must be available for use by the macro. It can also be used to supply the size of the field to be scanned.
- The global fields must be defined and an application register (other than R1, R2) must be loaded with the global area base address by the GLOBZ macro.

Return Conditions

- Control is returned to the next sequential instruction.

ALPHA

- The condition code can be changed by this macro. If it is nonzero, the scan specified was successful.
- If the condition code is nonzero, the address of the byte that satisfied the scan is provided. If the system has 16Mb or less of main storage, the address is returned in the low order 24 bits of R1. The high order 8 bits of R1 are unchanged. If the system has more than 16Mb of main storage, all 4 bytes contain the address.
- The high order 24 bits of R2 are unchanged by this macro.
- If the scan was successful, and if the option STOP=YES was specified or implied, the low order 8 bits of R2 contain the alpha group number of the character found expressed in binary format.
- If the scan was successful, and if the option STOP=NOT was specified, the low order 8 bits of R2 contain the value of one if the nonalphabetic character found was numeric (0–9) or a value greater than 1 if a special character was found. The value is expressed as a binary number.

Programming Considerations

- This macro can be run on any I-stream.
- If STOP=YES, this macro uses the global field @TRTAL-TRT Table, Alpha characters.
If STOP=NOT, the global field @TRTNO-TRT Table, non-Alpha characters, is used.

Do not use R0 because it is used in the EX instruction.

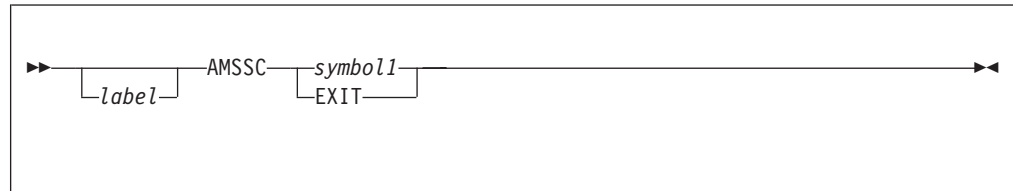
Examples

None.

AMSSC–Error Recording

This general macro is used to write an error record to SYS1.LOGREC for subsequent processing by the Environmental Error Record Editing and Printing Program (EREP).

Format



label

A symbolic name can be assigned to the macro statement.

symbol1

The label of the routine in the current program to which control will be passed if the recording is successful.

EXIT

The macro is to exit after successful recording.

Entry Requirements

Before issuing the macro, do the following:

- Initialize register 0 (R0) to the two's complement of the record length.
- Initialize R1 to the address of the record.

Return Conditions

- If recording is unsuccessful, control is returned to the instruction following the macro call.
- If recording is successful, control is passed to the routine specified as *symbol1*, or the EXITC macro is issued if the EXIT parameter was coded.
- 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 executed only in the main I-stream.
- If TPF is running under the control of VM/XA (*), the macro call will be trapped, the error record written to SYS1.LOGREC, and control passed to the routine specified by *symbol1*.
- If TPF is running native or if VM/XA fails to write the record, the macro call will be handled by TPF and control will be passed to the next sequential instruction plus 4 after the macro call. Then it is the programmer's responsibility to write the record to the real-time dump tape.

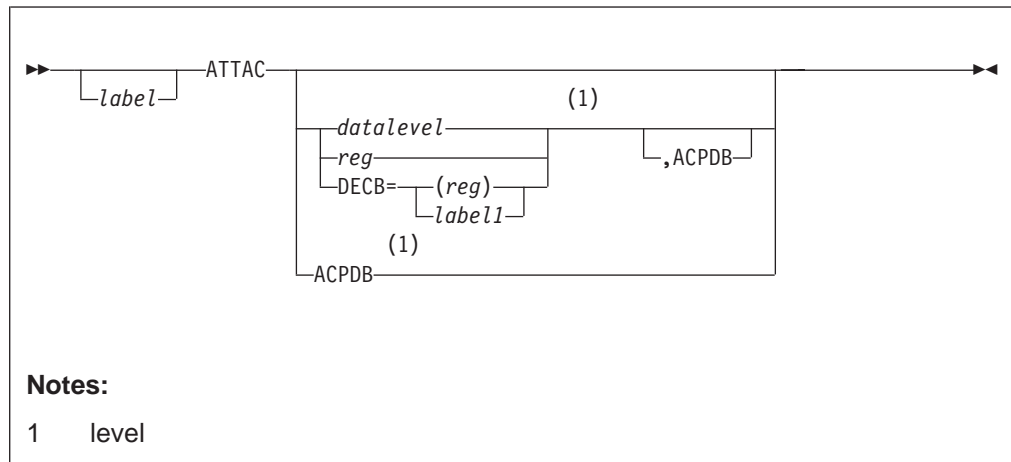
Examples

None.

ATTAC—Attach a Detached Working Storage Block

Use this general macro to reattach a working storage block to an entry control block (ECB) data level or data event control block (DECB). This storage block must have been previously detached by using the DETAC macro.

Format



label

A symbolic name can be assigned to the macro statement.

level

The ECB data level to which the block is to be attached. This can be specified in one of the following ways:

datalevel

Valid values range from D0 to DF.

reg

A register containing the ECB data level.

DECB=(reg)||label1

The label or general register (R1–R7) containing the address of the DECB where the block will be attached.

ACPDB

The user is TPFDF and the ECB data level is in register 7 (R7) if *level* or DECB is not specified.

Entry Requirements

- R9 must contain the address of the ECB being processed.
- A storage block must have been previously released from the same ECB data level or DECB before using the DETAC macro.
- The ECB must not be holding a storage block on the specified ECB data level or DECB.
- When ACPDB is specified, a unique identifier must be supplied in the file address reference word (FARW) of the ECB data level or DECB indicated. This identifier will be used when reattaching the block and will allow a user to control the order of reattachment.

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.
- If an ECB data level is coded as a register and that register is not R7, the contents of the specified register will be loaded into R7.
- The specified ECB data level or DECB:
 - Core block reference word (CBRW)
 - FARW
 - FARW extension will be restored to the contents at the time the most recent DETAC macro was issued for that level or DECB.

Programming Considerations

- This macro can be run on any I-stream.
- A block must have been released from the specified ECB data level or DECB by using the DETAC macro.
- The specified ECB data level or DECB must not be holding a block at the time the ATTAC macro is issued.
- When the ECB data level is not explicitly given or provided in a register, the level is assumed to reside in R7. Therefore, for ATTAC and ATTAC ACPDB, R7 contains the level.
For ATTAC *reg*, register *reg* contains the ECB data level.
For ATTAC *datalevel*, the ECB level is *datalevel*.
- The DETAC/ATTAC macros **push** and **pop** working storage blocks to and from a list chained from the ECB. ATTAC will always attach the block most recently detached from the specified ECB data level or DECB unless the ACPDB option is used (either as the first or second parameter).
- The ACPDB option allows you to control the order in which blocks are reattached to a given ECB data level or DECB.
 - A user indicates the block to be attached by specifying the ECB data level or DECB and supplying the identifier in the associated FARW.
 - The ATTAC service routine finds the block by searching the detached block table for a matching ECB data level or DECB and FARW identifier.

Therefore, ATTAC with ACPDB as the first positional parameter picks the ECB data level from R7 and uses data from the FARW corresponding to that level. The detached block list for that ECB data level is searched for the key given in the FARW and reattaches the detached block on the specified level. Similarly, ATTAC *reg*,ACPDB picks the ECB data level from *reg* and ATTAC *datalevel*,ACPDB uses ECB data level *datalevel* and its corresponding FARW. ATTAC DECB=(*reg*),ACPDB gets the DECB address from *reg* and uses the FARW in the DECB.

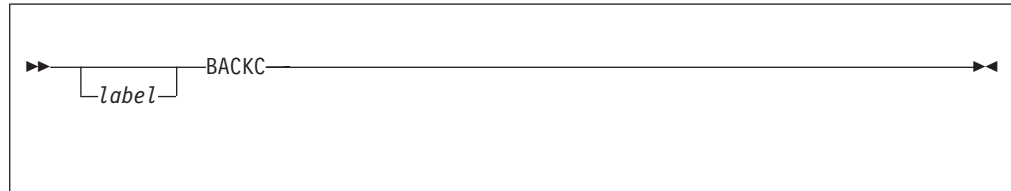
Examples

None.

BACKC—Return to Previous Program Record

This general macro returns control to the last program held by the entry control block (ECB) that issued an ENTRC.

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 never returned to the program that issues the BACKC macro.

Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB being processed before issuing this macro.
- The program that issued the last ENTRC macro will regain control in the addressing mode in which it was operating prior to issuing the ENTRC macro. The operational program registers R0–R7 have the same value they had when the BACKC macro was issued. The condition code and the contents of registers R14 and R15 are unpredictable.
- Following use of this macro, control can be transferred to the current program for processing of another message. Accordingly, it may be necessary to save and reset temporary counters and switches to allow for proper execution of this program.
- An ENTDC macro must not be issued between the execution of an ENTRC macro and a BACKC macro. If this sequence occurs, control is transferred to the system error routine.
- Users of the ALASC macro should note the programming considerations in the ALASC specifications relative to the BACKC macro.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.
- BACKC cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES).

Examples

None.

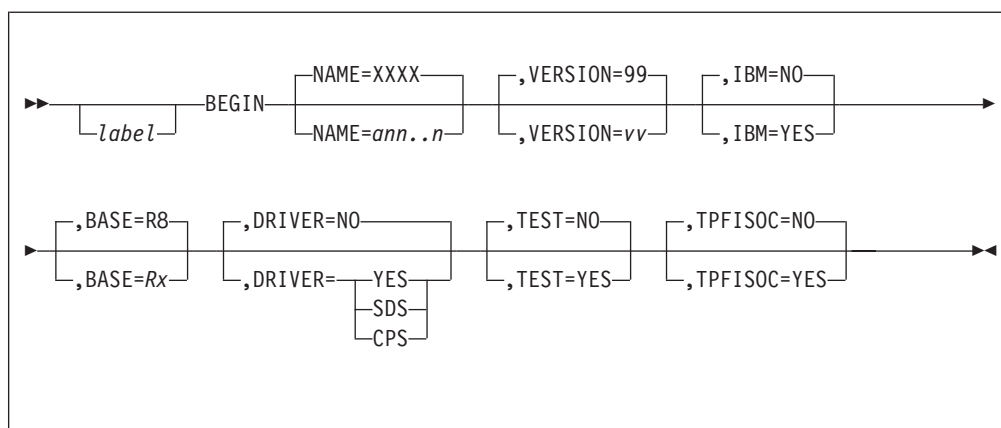
BEGIN—Begin Assembler Program

This general macro comes at the beginning of a TPF real-time program. It provides the necessary formatting for a program header in E-type assembler language programs. The information formatted by the macro is used by the system loader and the application core load and restart information. See *TPF System Installation Support Reference* for more information about the system loader and global area program.

BEGIN also generates a TITLE card with an operand field consisting of the NAME parameter concatenated with the VERSION parameter. BEGIN must be the first statement of each assembler language ECB-controlled program, including drivers, to require the use of the program CSECT because it defines the CSECT and program header.

BEGIN can also be coded in assembler code that is linked into ISO-C object files.

Format



label

A symbolic name may be assigned to the macro statement.

NAME=ann..n

The program name. The first character must be alphabetic; the remaining characters must be alphanumeric.

- If TPFISOC=NO is specified (or defaults), the program name must be 4 characters.
- If TPFISOC=YES is specified, the program name can be up to 6 characters. If the program name is a dynamic link module (DLM) entry point, it must match the 4-character DLM load module name.

The program name will be set to XXXX if not coded or not valid.

VERSION=vv

A 2-character version number to be associated with this program. *If not coded, a default version of 99 will be supplied.*

IBM

Specify one of the following:

NO

The default; signifies this is not IBM-released code.

BEGIN

YES

Signifies this is IBM-released code.

BASE=Rx

The base register to be used by this program. R8, the default, is assumed to be used by all nondriver programs. If TPFISOC=YES is specified, the BASE parameter default is R15.

DRIVER

Specify one of the following:

NO

The default, the program is not a driver.

YES

The program is a driver. R11 will be the base register.

SDS

This is used for drivers that do not want to activate the global restart programs CIJH and GOGO.

CPS

This is the same as in the YES option, but R8 will be the base register.

Do not code the DRIVER parameter when TPFISOC=YES is specified.

TEST

Specify one of the following:

NO

The default; indicates the TSTEQ equates will not be called.

YES

Indicates a request for the TSTEQ equates.

Do not code the TEST parameter when TPFISOC=YES is specified.

Entry Requirements

None.

Return Conditions

None.

Programming Considerations

- The normal program header is 8 bytes (2-byte record ID, 2-byte program length, 4-byte program name).
- A check is made by the macro for a valid NAME keyword. If the name is incorrect, the macro will insert the name XXXX. The program cannot be loaded with an XXXX name entry.
- This macro calls the following macros:
 - TSTEQ, when TEST=YES or DRIVER=YES|SDS|CPS
 - EB0EB, always
 - SYSEQ, always
 - SYSEQC, when IBM=NO
 - UXTEQ, when IBM=NO
 - XMSEQ, always.

- This macro will enter the global restart programs CIJH and GOGO on initial activation of this program when DRIVER is YES or CPS. In these cases the program is non-reentrant because it modifies itself to do this.
- When assembly language segments are linked into ISO-C object files, the TPFISOC parameter must be coded as YES. The BEGIN macro does not generate a program header, but does generate an ENTRY point label using the NAME parameter. BEGIN in ISO-C code is particularly useful for migrating TARGET(TPF) library functions; see also “TMSPC—Prolog for ISO-C Functions Calling TPF Macro Services” on page 401 and “TMSEC—Epilog for ISO-C Functions Calling TPF Macro Services” on page 399.

Examples

None.

BPKDC–TPF Input Message Tokenization Support

This general macro is a standardized TPF function for tokenizing input commands. It allows the command support routine to define to the tokenizer the expected format and values of an input message in a standardized way. This allows all input messages to be processed in a standard way and simplifies the operator interface and user factors.

A standard input command has 'Z' as the first character followed immediately by a blank or a 1- to 4-character secondary action code. The secondary action code either ends the message or is followed by a blank. For more information about the standard format of an input command see *TPF Programming Standards*.

The BPKDC macro is used in conjunction with the BPPSC macro to define the input message format. (See “BPPSC–Static Positional List Build” on page 56 for additional information.)

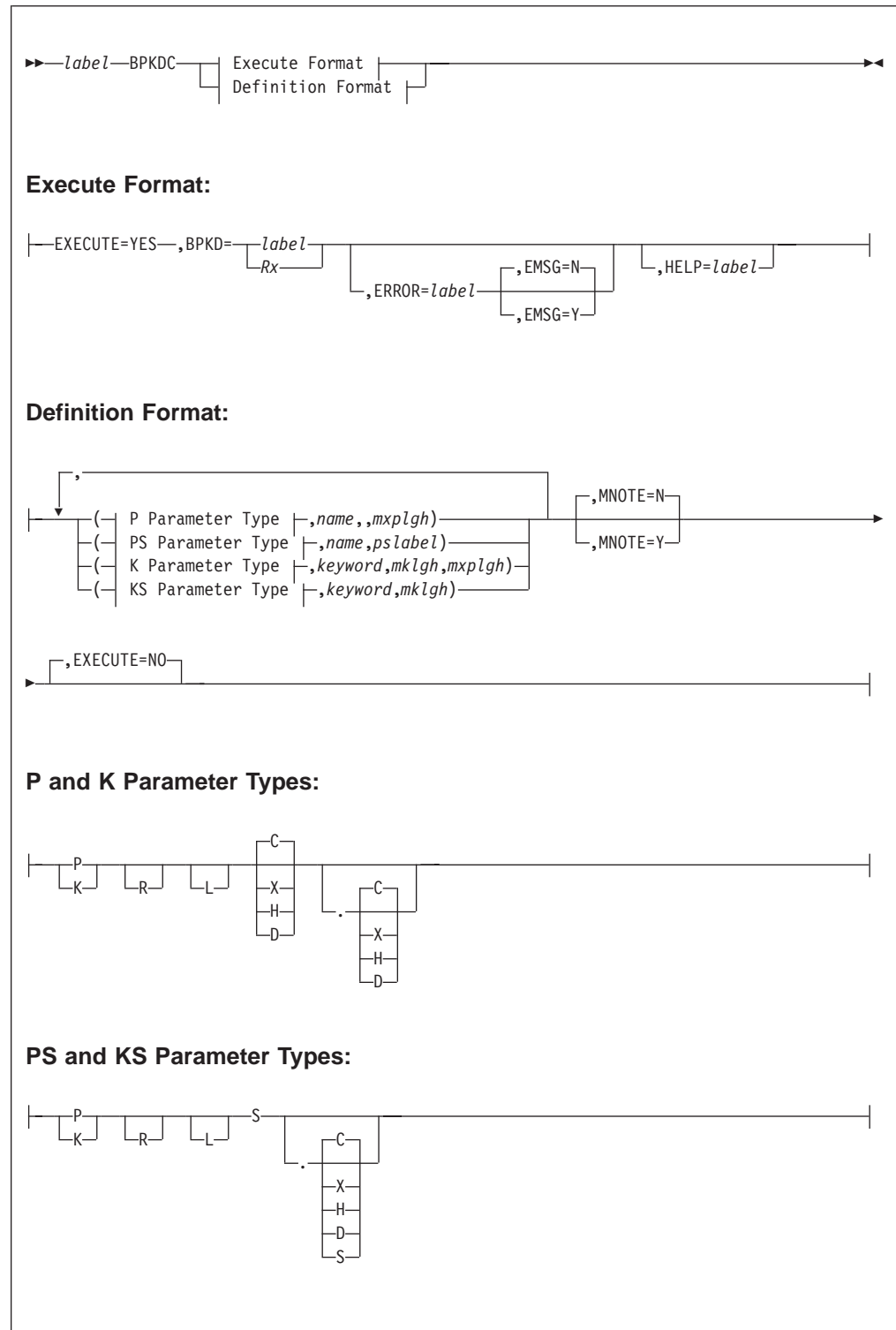
To define the input message format, the message processor program issues two macros that build the tables required by the tokenizer program. The definition form of BPKDC is used to define the actual message input format. The macro BPPSC is used to define allowable values for a specific positional parameter.

When the input message processor is given control, it invokes the tokenizer program by issuing the execute form of the BPKDC macro. The tokenizer builds an output parameter list that contains pointers to each of the tokens found in the input message. The sequence of output pointers is determined by the sequence of definitions in the input BPKDC. Also, if requested, the tokenizer performs some simple conversion of input parameters to usable binary output values.

When an error is encountered, the action taken by the tokenizer will depend upon the options specified on the BPKDC macro. One of the following actions will occur:

- Send an error response to the inputting terminal and exit the ECB
- Send an error response and return to the calling program
- Bypass sending an error response and return to the calling program.

Format



label

A symbolic name can be assigned to the macro statement.

EXECUTE=YES

This parameter is used to define the form of the BPKDC macro being specified.

BPKDC

If EXECUTE=YES, then the macro expands into a invocation of the BPKDC tokenizer program. If this is not an execute form, then the execute keyword should not be specified.

BPKD=*label*/*Rx*

The label of the definition BPKDC to be used to tokenize the input message on level 0. It may be specified as either a label or a register parameter.

ERROR=*label*

An optional parameter to indicate where to branch to if an error occurs in parsing the input. If it is not entered, then the tokenizer program will exit the ECB with an appropriate error message to the inputting terminal.

EMSG=Y|N

An optional parameter, which can be coded only with the ERROR parameter, to indicate whether an error will cause an error message to be issued to the inputting terminal before returning to the calling segment. The default is EMSG=N.

HELP=*label*

An optional parameter to indicate where to branch if the word 'HELP' or the character '?' appears as the first parameter.

type

A set of letters which define the parameter type and conversion to be performed. The following type codes are valid:

- P - positional parameter
- K - keyword parameter
- S - self-defining parameter
- R - parameter is required
- L - conversion character that follows is in the long (extended) format
- C - character string
- H - hexadecimal number is to be converted to binary

Maximum input value length is 8. The input will be treated as a number which can contain an odd number of hex characters and will on output be right-justified in the output field.

When specified with the L type code, the maximum input value length is 16. The input will be handled as a number that can contain an odd number of hexadecimal characters and will, on output, be right-justified in the output field.

- D - decimal number to be converted to binary

Maximum input value length is 8. The input will be treated as a number and will on output be right-justified in the output field.

- X - hexadecimal character string is to be converted to binary

Maximum input value length is 254. The input will be treated as a hexadecimal character string, which must contain an even number of hexadecimal characters.

- Not X, H, D, C or the period formats.

Maximum input value length is 255. The input will be treated as a character string.

The following rules about usage of the type codes apply:

Either P or K must be specified.
 P and K are mutually exclusive.
 L is valid only when H is specified.
 X, H, D, S, and C are mutually exclusive.

Note: The parameter definition fields are determined by the type code assigned to the parameter. The following are the legal field definitions by type codes:

P - (type,name,,mxplgh)
 PS - (type,name,pslabel)
 K - (type,keyword,mklgh,mxplgh)
 KS - (type,keyword,mklgh)

name

This is the name of the parameter as represented in the operator's guide. This name is used as a parameter for any error messages issued for this parameter. It must consist of 14 or fewer alphanumeric characters.

keyword

This is the keyword name as represented in the operator's guide. The value specified is the value that will be accepted as the keyword by the tokenizer. It must consist of 14 or fewer alphanumeric characters. This name is also used as a parameter for any error messages issued for this parameter.

mklgh

This is the minimum length acceptable for the name of a keyword parameter. This value must be less than the length of the keyword name.

pslabel

This is the name of a BPPSC macro which defines the accepted terms for a PS (self-defining positional) type of parameter.

mxplgh

This is the maximum input length for a legal parameter value.

MNOTE

This specifies whether BPKDC is to generate MNOTE messages describing the output DSECT label to be used to access each specified input parameter. The default is N.

Y Generate MNOTE messages.

N Do not generate MNOTE messages.

Entry Requirements

The tokenizer program expects data level 0 to contain the input command block in a MI0MI format. It also expects data level 1 and the extended switch EBXSW0 to be available for use.

Return Conditions

- The output of the tokenizer program is a pointer in R1 to the BPKDC output parameter list as defined in DCTBPK. The original message block from level 0 has been released and a new 4K block will reside on level 0. The output parameter list is contained in the 4K block.
- If an error occurs during the macro execution and the ERROR parameter has been entered, then a branch will be taken to that label in the calling segment. Data level 0 will still contain the input command block and Data level 1 will be available for use.

BPKDC

- If the HELP parameter is coded, a branch will be taken to that label in the calling segment. The original message block will be returned on D0 and there is no output parameter list generated.
- R0-R7, R14 and R15 are considered to be work registers and are not saved or restored over the call to the tokenizer. The tokenizer uses the register save area CE1UR0-CE1URB as a work area.
- The extended switch EBXSW0 has been cleared.

Programming Considerations

- This macro can be executed on any I-stream.
- Because of character restrictions on some terminals, the dollar sign character (\$) can be used in place of the quote character ('). Using a single \$ in a string results in the \$ being translated into a quote mark. To use \$ without translation two must be used together: \$\$.

The string \$This is a string\$ translates into 'This is a string'.

The string \$This is a \$\$string\$ translates into 'This is a \$string'.

- The following is a list of input command syntax rules for the BPKDC macro:
 - BPKDC syntax assigns the following characters special syntax definitions:

- ' ' (**blank**)
Token delimiter
 - ',' (**comma**)
Token delimiter
 - '/' (**slash**)
Token delimiter
 - '.' (**period**)
Special delimiter
 - '-' (**hyphen**)
Keyword character definition
 - '=' (**equal**)
Keyword character definition
 - '"' (**quote**)
Quoted character string delimiter

- #EOM
End of input message character

- "" Double quotes, result in single quote in output

- BPKDC syntax recognizes two types of parameters:
 - Positional parameters are those that must be entered in a specific position of an input message.
ZDSMG ACTION SDA ...
 - ACTION is positional and must always be the first parameter.
 - SDA is positional and must always be the second parameter.
 - Keyword parameters are those whose value is determined by having the keyword name assigned to the value. Keyword parameters can be entered in any order as long as they are entered after all positional parameters.
ZSIPC ALTER INTERVAL TIME-xx PRIM=ccc ALTERN-xx
 - ALTER is positional and must always be the first parameter
 - INTERVAL is positional and must always be the second parameter

- TIME is a keyword and can be entered in any position other than the first or second.
- PRIM is a keyword and can be entered in any position other than the first or second.

ZSIPC TIME-xx PRIM-xxx ALTER INTERVAL

The above is invalid because the keywords were entered before the required positional parameters.

- Parameters are delimited by either a blank, a comma, or a slash. Multiple delimiters between parameters will be deleted. Since TPF syntax does not currently support a null positional parameter, all multiple delimiters will be deleted.

A B,C	contains three parameters A, B and C
A , B	contains two parameters A and B
A ,, B	contains parameters A and B and the null parameter is ignored.

- The period (.) is a special delimiter that ties two parameters together. (See type definitions.)
- All leading and trailing blanks will be deleted from each parameter.
- A keyword is an alphanumeric character string of 14 characters or less, and it can end with a *single* occurrence of a '-' or a '='.
- There are two forms of keywords:
 - Keywords that require a value
 - Keywords that are self-defining and require no additional parameter.
- Keywords that require a value are entered in the following form:

```
Keyword name= value
      or
Keyword name- value
```

There can be **no** intervening blanks between the keyword and the character '-' or '='. There can be blanks between the character '-' or '=' and the value to be assigned to the keyword. The following are valid keyword parameters:

```
KEY-aaaa
KEY- aaaaa
```

The following are not valid:

```
KEY - xxxxxxx blank between KEY and '-'
KEY -xxxxxxx blank between KEY and '-'
KEY--xxxxxxx multiple occurrence of '-'.
```

A null value can be entered for a keyword by entering the keyword followed by a comma:

```
KEY- ,      comma causes null value for keyword
```

- Self-defining keywords can take one of only two values: a YES value, which is the keyword name; or a NO value, which is the keyword name preceded by the characters 'NO'.
- ```
keyword is ERASE
if ERASE entered, then value assigned is 'Y'
if NOERASE entered, then value assigned is 'N'
```
- If a keyword is entered more than once, then the value assigned is the last entered value.

```
KEY-A KEY-DD DD is the value assigned to KEY.
KEY-A, KEY-, KEY will be assigned a null value
```

## BPKDC

- Quoted character strings are character strings that start with a quotation mark and end with a quotation mark. Quoted character strings can contain any character except the #EOM character or a single quote. If a quote is to be entered as an input character, it must be doubled. The dollar sign (\$) character can be used as an alternative to single quotes for terminals that do not support the single quote.

```
'adada=adad=' is a valid quoted character string
'adadad'afafa' is not a valid character string
'adadad''afafa' is a valid character string which
contains single quote as part of the string.
```

- The output parameter list (DCTBPK) consists of a series of fullwords that will contain either 0 (no parameter entered), an index value (for self-defining positional parameters) or an address, which points to either a fullword value containing the converted input parameter, a character string preceded by a 1-byte length field, or a doubleword, where each fullword contains an address which points to either a fullword value or a character string preceded by a length byte. The form of the output parameter returned is determined by the type of parameter and whether conversion was requested.

The number of output fullwords is always two more than the number of defined parameters in the BPKDC used to tokenize the input message. The tokenizer places a count of output parameters in the first word of the output parameter list. The second word always points to the 4-character Z message secondary action code. That is, ZPROC the second word, would point to the 4-character string 'PROC'. The next word would start the parameters in the sequence defined in the BPKDC macro.

Therefore, for the BPKDC definition:

```
BPKD1 BPKDC (PSR,action,list1),(PSR,subject,list2),(KDR,pid,,2)
LIST1 BPPSC (DISPLAY,1),(ALTER,1)
LIST2 BPPSC (MASS,1),(STATUS,1)
```

The output parameter list for the following command:

```
ZPROC D S PID=01
```

would look like the following:

| PLIST | Description                   | Value           |
|-------|-------------------------------|-----------------|
| word1 | - count of parameters         | - 00000004      |
| word2 | - address of character string | - x'04',c'PROC' |
| word3 | - index to Display in LIST1   | - 00000004      |
| word4 | - index to Status in LIST2    | - 00000008      |
| word5 | - address of fullword set to  | - 00000001      |

Where:

- Word1 is the count of output parameters being returned by the tokenizer.
- Word2 is the address of the character string defining the input message suffix, which in this case is c'PROC'.
- Word3 contains the index value of the valid parameter specified for position 1 and found in the self-defining positional list built using the BPPSC macro. If the action word value had been 'ALTER' the index value would have been an '8'.
- Word4 contains the index value of the valid parameter specified for position 2 and found in the self-defining positional list built using the BPPSC macro. If the subject word value had been 'MASS' the index value would have been a '4'.

- Word5 contains the address of a fullword that contains the binary value of the converted decimal input parameter specified for keyword 'PID'.
- The following type codes define the type of input data expected. A parameter is defined as either a positional (Type code=P) or as a keyword (Type code=K) parameter. The following are valid for both keywords and positionals:

- C** The output value for a type 'C' is a pointer to a character string, where the first byte of the string is the length of the string.

word= address -> lcccccccccccc (character string)

**Note:** The character type is the default type. If no other type code is specified, BPKDC will expect a character string.

- H** The output value for a type 'H' is a pointer to a fullword containing the binary equivalent of the entered hexadecimal number, right-justified.

word= address -> hhhhhhhh

#### HL

The output value for a type 'HL' is a pointer to a doubleword containing the binary equivalent of the entered hexadecimal number, right-justified.

word= address -> hhhhhhhhhhhhhhhh

- D** The output value for a type 'D' is a pointer to a fullword containing the binary equivalent of the entered decimal number, right-justified.

word= address -> dddddddd

- X** The output value for a type 'X' is a pointer to a character string containing the binary equivalent of the entered hexadecimal character string preceded by one byte containing the length of the string. The input string must contain an even number of characters.

word= address -> lxxxxxxxxxxxxxx

#### C.H

The output value for a type 'C.H' is a pointer to a doubleword, where the first fullword contains a pointer to a character string where the first byte of the string is the length of the string. The second fullword contains a pointer to a fullword containing the binary equivalent of the hexadecimal number, right justified.

word= address-> address -> lccccccc (character string)  
address -> hhhhhhhh

**Note:** This type can handle the following input formats:

- chr1.hex2, where chr1 is a character string and hex2 is a hexadecimal number
- chr1. or chr1
- .hex2

**Note:** If one of the parameters is not entered, then it's corresponding fullword in the doubleword will contain zeros.

#### H.H

The output value for a type 'H.H' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the second hexadecimal number, right justified.

word= address-> address -> hhhhhhhh  
address -> hhhhhhhh

**Note:** This type can handle the following input formats:

- hex1.hex2, where hex1 and hex2 are two hexadecimal numbers
- hex1. or hex1
- .hex2

### HL.H

The output value for a type 'HL.H' is a pointer to a doubleword where, the first fullword contains a pointer to a doubleword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the second hexadecimal number, right justified.

```
word= address-> address -> hhhhhhhhhhhhhhhh
 address -> hhhhhhhh
```

**Note:** This type can handle the following input formats:

- hex1.hex2, where hex1 and hex2 are two hexadecimal numbers
- hex1. or hex1
- .hex2

### D.H

The output value for a type 'D.H' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the hexadecimal number, right justified.

```
word= address-> address -> dddddddd
 address -> hhhhhhhh
```

**Note:** This type can handle the following input formats:

- dec1.hex2, where dec1 is a decimal number and hex2 is a hexadecimal number
- dec1. or dec1
- .hex2

### X.H

The output value for a type 'X.H' is a pointer to a doubleword, where the first fullword contains a pointer to a character string containing the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string. The second fullword contains a pointer to a fullword containing the binary equivalent of the hexadecimal number, right justified.

```
word= address-> address -> 1xxxxxxx
 address -> hhhhhhhh
```

**Note:** This type can handle the following input formats:

- xxx1.hex2, where xxx1 is a hexadecimal character string and hex2 is a hexadecimal number
- xxx1. or xxx1
- .hex2

### C.D

The output value for a type 'C.D' is a pointer to a doubleword, where the first fullword contains a pointer to a character string where the first byte of the string is the length of the string. The second fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified.



```
word= address-> address -> lccccccc (character string)
 address -> dddddddd
```

**Note:** This type can handle the following input formats:

- chr1.dec2, where chr1 is a character string and dec2 is a decimal number
- chr1. or chr1
- .dec2

#### H.D

The output value for a type 'H.D' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified.

```
word= address-> address -> hhhhhhhh
 address -> dddddddd
```

**Note:** This type can handle the following input formats:

- hex1.dec2, where hex1 is a hexadecimal number and dec2 is a decimal number
- hex1. or hex1
- .dec2

#### HL.D

The output value for a type 'HL.D' is a pointer to a doubleword, where the first fullword contains a pointer to a doubleword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified.

```
word= address-> address -> hhhhhhhhhhhhhhhh
 address -> dddddddd
```

**Note:** This type can handle the following input formats:

- hex1.dec2, where hex1 is a hexadecimal number and dec2 is a decimal number
- hex1. or hex1
- .dec2

#### D.D

The output value for a type 'D.D' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the first decimal number, right justified. The second fullword contains a pointer to a fullword containing the binary equivalent of the second decimal number, right justified.

```
word= address-> address -> dddddddd
 address -> dddddddd
```

**Note:** This type can handle the following input formats:

- dec1.dec2, where dec1 and dec2 are two decimal numbers
- dec1. or dec1
- .dec2

#### X.D

The output value for a type 'X.D' is a pointer to a doubleword, where the first

## BPKDC

fullword contains a pointer to a character string containing the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string. The second fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified.

```
word= address-> address -> 1xxxxxxx
 address -> dddddddd
```

**Note:** This type can handle the following input formats:

- xxx1.dec2, where xxx1 is a hexadecimal character string and dec2 is a decimal number
- xxx1. or xxx1
- .dec2

## C.X

The output value for a type 'C.X' is a pointer to a doubleword, where the first fullword contains a pointer to a character string where the first byte of the string is the length of the string. The second fullword contains a pointer to a character string that contains the binary equivalent of the entered hexadecimal string, where the first byte is the length of the string.

```
word= address-> address -> lccccccc (character string)
 address -> 1xxxxxx
```

**Note:** This type can handle the following input formats:

- chr1.xxx2, where chr1 is a character string and xxx2 is a hexadecimal string.
- chr1. or chr1
- .xxx2

## H.X

The output value for a type 'H.X' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the entered hexadecimal number, right justified. The second fullword contains a pointer to a character string that contains the binary equivalent of the entered hexadecimal string, where the first byte is the length of the string.

```
word= address-> address -> hhhhhhhh
 address -> 1xxxxxx
```

**Note:** This type can handle the following input formats:

- hex1.xxx2, where hex1 is a hexadecimal number and xxx2 is a hexadecimal string.
- hex1. or hex1
- .xxx2

## HL.X

The output value for a type 'HL.X' is a pointer to a doubleword, where the first fullword contains a pointer to a doubleword containing the binary equivalent of the entered hexadecimal number, right justified. The second fullword contains a pointer to a character string that contains the binary equivalent of the entered hexadecimal string, where the first byte is the length of the string.

```
word= address-> address -> hhhhhhhhhhhhhhhh
 address -> 1xxxxxx
```

**Note:** This type can handle the following input formats:

- hex1.xxx2, where hex1 is a hexadecimal number and xxx2 is a hexadecimal string
- hex1. or hex1
- .xxx2

**D.X**

The output value for a type 'D.X' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the entered decimal number, right justified. The second fullword contains a pointer to a character string that contains the binary equivalent of the entered hexadecimal string, where the first byte is the length of the string.

```
word= address-> address -> dddddddd
 address -> 1xxxxxxx
```

**Note:** This type can handle the following input formats:

- dec1.xxx2, where dec1 is a decimal number and xxx2 is a hexadecimal string.
- dec1. or dec1
- .xxx2

**X.X**

The output value for a type 'X.X' is a pointer to a doubleword, where the first fullword contains a pointer to a character string containing the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string. The second fullword contains a pointer to a character string containing the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string.

```
word= address-> address -> 1xxxxxxxxx
 address -> 1xxxxxxx
```

**Note:** This type can handle the following input formats:

- xxx1.xxx2, where xxx1,xxx2 are hexadecimal character strings
- xxx1. or xxx1
- .xxx2

**C.C**

The output value for a type 'C.C' is a pointer to a doubleword, where the first fullword contains a pointer to a character string where the first byte of the string is the length of the string. The second fullword contains a pointer to a character string where the first byte is the length of the string.

```
word= address-> address -> lccccccc (character string)
 address -> lcccccc (character string)
```

**Note:** This type can handle the following input formats:

- chr1.chr2, where chr1,chr2 are character strings
- chr1. or chr1
- .chr2

**H.C**

The output value for a type 'H.C' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the entered hexadecimal number, right justified. The second fullword contains a pointer to a character string, where the first byte is the length of the string.

```
word= address-> address -> hhhhhhhh
 address -> lccccccc
```

**Note:** This type can handle the following input formats:

- hex1.chr2, where hex1 is a hexadecimal number and chr2 is a character string.
- hex1. or hex1
- .chr2

### HL.C

The output value for a type 'HL.C' is a pointer to a doubleword, where the first fullword contains a pointer to a doubleword containing the binary equivalent of the entered hexadecimal number, right justified. The second fullword contains a pointer to a character string, where the first byte is the length of the string.

```
word= address-> address -> hhhhhhhhhhhhhh
 address -> lcccccc
```

**Note:** This type can handle the following input formats:

- hex1.chr2, where hex1 is a hexadecimal number and chr2 is a character string.
- hex1. or hex1
- .chr2

### D.C

The output value for a type 'D.C' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the entered decimal number, right justified. The second fullword contains a pointer to a character string, where the first byte is the length of the string.

```
word= address-> address -> dddddddd
 address -> lcccccc
```

**Note:** This type can handle the following input formats:

- dec1.chr2, where dec1 is a decimal number and chr2 is a character string.
- dec1. or dec1
- .chr2

### X.C

The output value for a type 'X.C' is a pointer to a doubleword, where the first fullword contains a pointer to a character string that contains the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string. The second fullword contains a pointer to a character string where the first byte is the length of the string.

```
word= address-> address -> lxxxxxxx
 address -> lcccccc (character string)
```

**Note:** This type can handle the following input formats:

- xxx1.chr2, where xxx1 is a hexadecimal string and chr2 is a character string
- xxx1. or xxx1
- .chr2

### C.HL

The output value for a type 'C.HL' is a pointer to a doubleword, where the first fullword contains a pointer to a character string where the first byte of

the string is the length of the string. The second fullword contains a pointer to a doubleword containing the binary equivalent of the hexadecimal number, right justified.

```
word= address-> address -> lccccccc (character string)
 address -> hhhhhhhhhhhhhhhh
```

**Note:** This type can handle the following input formats:

- chr1.hex2, where chr1 is a character string and hex2 is a hexadecimal number
- chr1. or chr1
- .hex2

## H.HL

The output value for a type 'H.HL' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a doubleword containing the binary equivalent of the second hexadecimal number, right justified.

```
word= address-> address -> hhhhhhhh
 address -> hhhhhhhhhhhhhhhh
```

**Note:** This type can handle the following input formats:

- hex1.hex2, where hex1 and hex2 are two hexadecimal numbers
- hex1. or hex1
- .hex2

## HL.HL

The output value for a type 'HL.HL' is a pointer to a doubleword, where the first fullword contains a pointer to a doubleword containing the binary equivalent of the first hexadecimal number, right justified. The second fullword contains a pointer to a doubleword containing the binary equivalent of the second hexadecimal number, right justified.

```
word= address-> address -> hhhhhhhhhhhhhhhh
 address -> hhhhhhhhhhhhhhhh
```

**Note:** This type can handle the following input formats:

- hex1.hex2, where hex1 and hex2 are two hexadecimal numbers
- hex1. or hex1
- .hex2

## D.HL

The output value for a type 'D.HL' is a pointer to a doubleword, where the first fullword contains a pointer to a fullword containing the binary equivalent of the decimal number, right justified. The second fullword contains a pointer to a doubleword containing the binary equivalent of the hexadecimal number, right justified.

```
word= address-> address -> dddddddd
 address -> hhhhhhhhhhhhhhhh
```

**Note:** This type can handle the following input formats:

- dec1.hex2, where dec1 is a decimal number and hex2 is a hexadecimal number
- dec1. or dec1
- .hex2

**X.HL**

The output value for a type 'X.HL' is a pointer to a doubleword, where the first fullword contains a pointer to a character string containing the binary equivalent of the entered hexadecimal number, where the first byte of the string is the length of the string. The second fullword contains a pointer to a doubleword containing the binary equivalent of the hexadecimal number, right justified.

```
word= address-> address -> 1xxxxxxx
 address -> hhhhhhhhhhhhhh
```

**Note:** This type can handle the following input formats:

- xxx1.hex2, where xxx1 is a hexadecimal character string and hex2 is a hexadecimal number
- xxx1. or xxx1
- .hex2

**Null**

If the parameter was not specified and was not a required parameter, its location in the output parameter list will contain a word of 0s.

**S** When specified with a keyword parameter (KS), the output value is a pointer to a one byte character string containing either the character 'Y' or 'N' depending on the input value. The character will be preceded by a length byte of value 1.

```
word= address -> xc (where c is either Y/N)
```

When specified with a positional parameter (PS), the output value is an index modular 4 where the index value is determined by the parameter's position in the list built by the specified BPPSC macro. If found, the index will always be equal to or greater than 4.

```
word= fullword index value 4 or greater.
```

The following are some examples of type codes and the expected input:

- PHR - The expected input is a required positional parameter which is a hexadecimal number.
- KD - The expected input is a keyword whose value will be a decimal number. It is not a required parameter.
- PX.X - the expected input is two hexadecimal strings, separated by a period. It is not a required parameter.
- KC.CR - the expected input is a keyword whose value is two character strings separated by a period. This parameter is required.

**Examples**

- Start message processing

```
SPACE 1
*
* FIRST BPKDC IS USED TO START MESSAGE PROCESSING AND WILL HANDLE
* A ZMCPY RESTART MESSAGE.
*
SPACE 1
MCPYBPKD BPKDC (PSR,TYPE,TYPEBPK2)
TYPEBPK2 BPPSC (ABORT,2,BPKD2),(RESTART,1),(ALL,,BPKD3)
SPACE 1
```

- ZMCPY ABORT handling

```

*
* THIS BPKDC WILL HANDLE A ZMCPY ABORT BP MESSAGE
*
 SPACE 1
BPKD2 BPKDC (PSR,TYPE,TYPEBPK2),(KS,BP)
 SPACE 1
• ZMCPY ALL handling
*
* THIS BPKDC WILL HANDLE A ZMCPY ALL XXXX TO XXXX MESSAGE
*
 SPACE 1
BPKD3 BPKDC (PSR,TYPE,TYPEBPK2), X
 (PHR,FROMSDA,,4), X
 (PSR,TO,TOBPK2), X
 (PHR,TOSDA,,4)
TOBPK2 BPPSC TO

```

The BPKDC 'MCPYBPKD' is used to start the input message processing. The first parameter expected is a self-defining positional, and the legal values are defined by the TYPEBPK2 BPPSC macro. If either 'ABORT' or 'ALL' is entered then a switch to another BPKDC takes place. 'ABORT' causes a switch to the BPKDC with label 'BPKD2' and 'ALL' causes a switch to the BPKDC 'BPKD3'. If 'RESTART' is entered then processing continues with this BPKDC. If the entered value is not one of these, the request is cancelled and either the ECB is exited or control is passed back to the calling segment depending on if the ERROR parameter was coded on the execution form.

By providing a switch capability, it is possible to restrict input parameters to those commands on which the input parameters are valid. In this example, ZMCPY ABORT BP, the BP keyword is only allowed when 'ABORT' has been specified. Therefore if it is entered on one of the other types of ZMCPY, it will not be accepted as valid because the BPKDCs for those messages do not have the 'BP' defined for them.

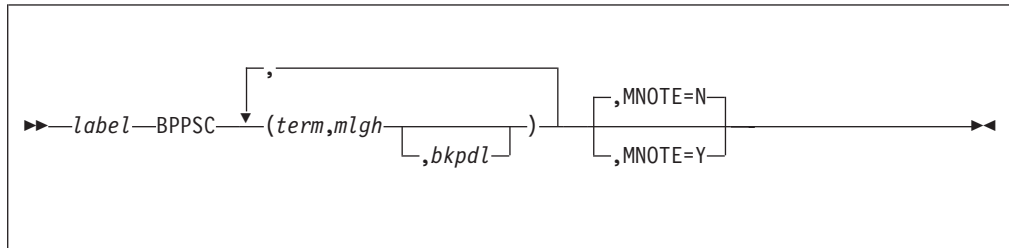
The 'ALL' case shows another reason for the BPKDC switch. It allows the input parameters to be defined as to the conversion to be performed on them. The 'FROMSDA' and the 'TOSDA' are entered as hexadecimal input, and this BPKDC specifies that they are to be converted to binary hexadecimal numbers. Also the 'TO' PS parameter allows the command to be tested for correct syntax. The 'TO' value can only be 'TO' (as defined in the BPPSC) and must be specified. If it is not entered, or not entered as 'TO' then an error message is written and the request is cancelled. This fulfills the syntax checking required by the operator's guide.

**Note:** When doing BPKDC switching as in the above paragraph, the BPKDC parameter definitions before the switch point must be the same in all the possible branch to BPKDCs. In other words, if the switch parameter is the third parameter, the first two parameter definitions must be the same in the BPKDC statements. In the above example the first parameter definition is repeated in each of the three BPKDCs. This is required because the tokenizer does not change input or output parameter displacements when it performs a BPKDC switch. It assumes that it will start processing the new BPKDC at the same displacement it would have continued with in the old BPKDC if no switch had occurred.

## BPPSC–Static Positional List Build

This general macro is used to define a list of valid self-defining terms allowed for a specific positional parameter. This macro is for use only in conjunction with the BPKDC macro. (See “BPKDC–TPF Input Message Tokenization Support” on page 40.)

### Format



*label*

A symbolic name must be assigned to the macro statement.

*term*

A legal value for this position in the input command.

*mlgh*

The minimum acceptable length for this term. This value must be less than length of the value for “term”.

*bkdI*

The label assigned to the BPKDC to be used if this term is entered. This parameter is used to allow the BPKDC processor to switch to a different BPKDC macro definition in order to allow for multiple input formats for the same command.

#### **MNOTE**

Specifies whether BPPSC is to generate MNOTE messages describing the index value for each specified input parameter. Default is N.

**Y** Generate MNOTE messages.

**N** Do not generate MNOTE messages.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

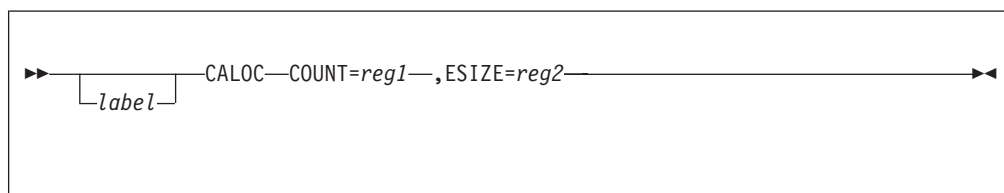
- This macro is usable only with the BPKDC macro. For further information about the function being performed, see the documentation for the BPKDC macro: “BPKDC–TPF Input Message Tokenization Support” on page 40.
- The BPPSC macro should only be used on the main I-stream.



## CALOC—Reserve and Initialize Storage

This general macro obtains a variable-sized, doubleword-aligned block from heap-resident storage and initializes it.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**COUNT=reg1**

A general register (R0-R7, R14, or R15) or a register equate containing the number of elements to be allocated.

**ESIZE=reg2**

A general register (R0-R7, R14, or R15) or equate containing the size of each element to be allocated.

### Entry Requirements

None.

### Return Conditions

- The register specified by COUNT contains the address of the start of the allocated storage. This storage is guaranteed to be doubleword aligned and initialized to hex zeros.

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

- The contents of the register specified by ESIZE are unaltered.
- The register specified for COUNT contains X'00000000' upon return if:
  - COUNT or ESIZE is zero.
  - The heap does not contain enough storage to satisfy the request for allocation, and the heap cannot be further extended.
- The contents of R14 and R15 are unknown. All other registers are preserved, except for any used for the COUNT parameter.
- Control is returned to the NSI if no heap corruption is detected.
- When main storage blocks are exhausted, a catastrophic system error results, since insufficient real storage exists to satisfy the request. Control is not returned.
- Control is not returned if corruption is detected in the heap. An error occurs and the ECB exits immediately. See *Messages (System Error and Offline)* and *Messages (Online)* for more details about this error.

### Programming Considerations

- Any valid register for an ECB-controlled program (R0-R7, R14, or R15) can be used for BLOCK or ESIZE.

## CALOC

- The amount of heap-resident storage available for use as ECB-unique storage is determined during installation. Check with the responsible party if the maximum is in doubt. This limit applies to the sum total of allocation requests issued by the ECB, not just to any single request.
- In addition to the normal macro trace information the macro trace entry will contain the size, in number of bytes, and address of the allocated storage.

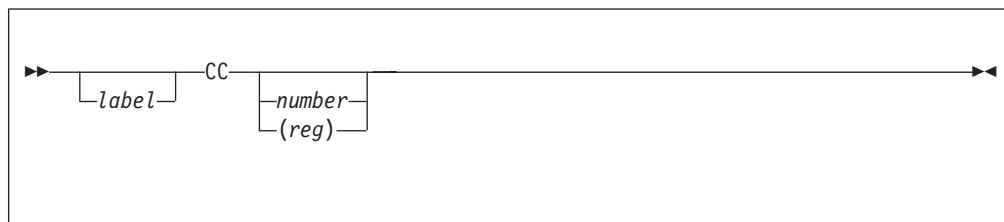
## Examples

None.

## CC–Set Condition Code in Current PSW

This general macro generates instructions to set the specified condition code in the current program status word (PSW). It is normally used by service routines that pass the condition code as part of their interface.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*number*

The condition code to be set (0, 1, 2, or 3).

*reg*

A register in which bits 30–31 contain the condition code to be set.

### Entry Requirements

A base register must be in effect for the macro.

### Return Conditions

The specified condition code has been set in the current PSW.

### Programming Considerations

- This macro can be executed on any I-stream.
- If no operand is specified, this macro generates equates that are useful in conditional branching instructions.

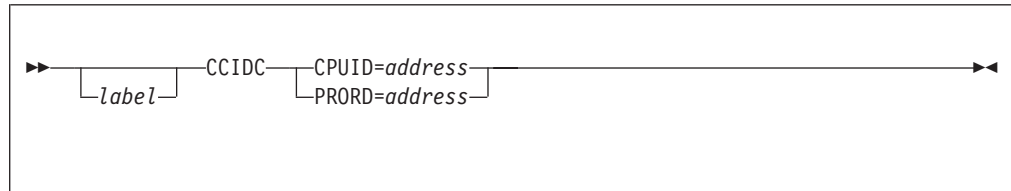
### Examples

None.

## CCIDC—Convert CPUID to Processor Ordinal Number

This general macro is used to convert a given CPUID (alpha) to its corresponding processor ordinal number, or vice versa. This macro is designed for use in a loosely coupled (LC) environment in which the Processor ID Table has been generated. Given the symbolic address of a 1-byte field containing the CPUID or the processor ordinal number, this macro will return the associated processor ordinal number or CPUID.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**CPUID=address**

The symbolic address of a one-byte field containing CPUID to be converted.

**PRORD=address**

The symbolic address of a one-byte field containing processor ordinal number to be converted.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction.
- Return code: R0 will contain:
  - 0 = successful completion
  - 4 = specified input is not in the PIDT table
- The contents of R14 is unpredictable.
- R15 will contain the requested PRORD (processor ordinal number) if the CPUID parameter has been coded.
- R15 will contain the requested CPUID if PRORD has been coded.
- The contents of all other registers are preserved across this macro call.

### Programming Considerations

- This macro can be executed on any I-stream.
- The CCIDC macro is for use in an LC environment. The macro is coded to function in a uniprocessor environment as well. In this environment, the macro will return a PRORD of zero and will use the CE1CPD field to establish the CPUID.
- Only one parameter should be coded. If both parameters are coded, the CPUID parameter will be used.

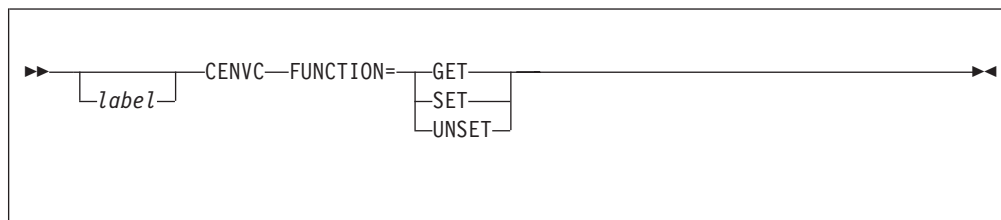
### Examples

None.

## CENVC—Access Environment Variables

This general macro is the assembler interface to the `getenv`, `setenv`, and `unsetenv` functions, which get and set the values of the environment list variables of a process, or remove variables from the environment list of a process.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### FUNCTION

This parameter specifies one of the following functions to perform on the environment list for the process:

#### GET

Searches the environment list for a variable name and returns a pointer to a string containing its associated value.

#### SET

Adds or changes the value of an environment variable. Only the private environment list of the process is changed; the global default environment list is not changed.

#### UNSET

Deletes an environment variable. Only the private environment list of the process is changed; the global default environment list is not changed.

### Entry Requirements

- Register R1 must contain the address of a parameter list. The format and contents of the parameter list depend on the value of the FUNCTION parameter:

#### GET

The parameter list consists of one fullword, that contains the address of a string containing the environment variable name. The string must not include any bytes containing the values `X'00'` or `C'='` and must end with a byte containing the value `X'00'`.

#### SET

The parameter list consists of three fullwords:

1. The address of a string containing the name of the environment variable to be added or changed. The string must not include any bytes containing the values `X'00'` or `C'='` and must end with a byte containing the value `X'00'`.
2. The address of a string containing the value to be associated with the name of the environment variable. The string must not include any bytes containing the value `X'00'` and must end with a byte containing the value `X'00'`.
3. A change flag. If the change flag contains the value 0 (`X'00000000'`), and the environment variable named by the first parameter already exists, its

value will not be changed. Otherwise (the change flag contains a nonzero value), the previous value of an existing environment variable will be changed to the new value given by the second parameter. If the environment variable does not already exist, it will be added to the environment list regardless of the value of the change flag.

#### UNSET

The parameter list consists of one fullword, which contains the address of a string containing the environment variable name. The string must not include any bytes containing the values X'00' or C'=' and must end with a byte containing the value X'00'.

- Callers from the control program (CP) must be in supervisor state, in 31-bit mode, operating in key 0, and in the ECB virtual memory (EVM).

## Return Conditions

- In the CP, the contents of register 7 (R7) will not be preserved across this macro call.
- The contents of R15 depend on the value of the FUNCTION parameter and the success or failure of the function that is being performed:

#### GET

If the environment variable exists, R15 contains the address of a string containing its value. The string does not include any bytes that contain the value X'00', and ends with a byte that contains the value X'00'. The ECB should not change the value of this string; any such modification will have results that cannot be predicted.

The program must be in 31-bit mode to view the data.

If the environment variable does not exist, R15 contains the value F'0'.

#### SET

If the requested SET function was performed, R15 contains the value F'0'. R15 contains the value F'-1' if the parameter was incorrect. R15 contains the value F'-2' if it was unable to obtain storage to obtain the environment list.

**Note:** The SET function is considered successful when the change flag is set to zero and prevents a change to the value of a preexisting environment variable. Possible reasons for failure include not having the ability to allocate memory for a new environment variable, or being passed a parameter list that is not valid.

#### UNSET

If the environment variable is removed from the environment list or was not present in the environment list, R15 contains the value F'0'. If the UNSET function is not successful (for example, the parameter list passed to it was not valid), R15 contains the value F'-1'.

- All other registers are unchanged.

## Programming Considerations

- The functions provided by this macro are equivalent to the getenv, setenv, and unsetenv C library functions.
- Environment variable names are case sensitive; that is, e and E name two different environment variables.
- This macro does not require that the ECB has previously initialized a C run-time environment and does not cause a C run-time environment to be created.

- If the ECB creates a new ECB by calling the CRESC macro or the system C library function, the new ECB inherits a *copy* of the environment list of the parent process. The child ECB cannot change the environment list for the parent process.
- The values of the following environment variables are used during initialization of the C run-time environment:
  - If called from the CP, this macro expects R13 to point to a stack.
  - This macro affects the environment list owned by the ECB that has control of the I-stream on which the CENVC macro is called.
  - Callers of this macro must be in the EVM.
  - Callers of this macro must be in 31-bit mode.

## Examples

```

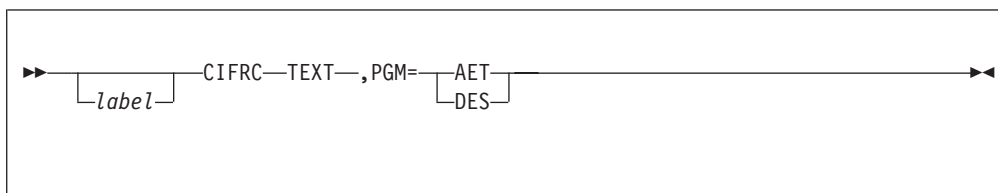
*
* Set an environment variable.
*
 LA R1,VARNAME Set the address of the name ...
 ST R1,PLIST as the 1st parameter.
 LA R1,SETVAL Set the address of the value ...
 ST R1,PLIST+4 as the 2nd parameter.
 L R1,CHNGFLG Set the change flag
 ST R1,PLIST+8 as the 3rd parameter.
 LA R1,PLIST Point R1 to the beginning of the plist.
 CENVC FUNCTION=SET Call CENVC to set the variable.
 LTR R15,R15 Was the SET successful?
 BNZ ERROR1 No, branch to error routine.
*
* Get the value of the environment variable that was just set.
*
 LA R1,VARNAME Set the address of the name ...
 ST R1,PLIST as the parameter.
 LA R1,PLIST Point R1 to the beginning of the plist.
 CENVC FUNCTION=GET Call CENVC to get the value
 LTR R15,R15 Was it found?
 BZ ERROR2 No, branch to error routine.
*
* Delete the environment variable.
*
 LA R1,VARNAME Set the address of the name ...
 ST R1,PLIST as the parameter.
 LA R1,PLIST Point R1 to the beginning of the plist.
 CENVC FUNCTION=UNSET Call CENVC to delete the variable.
 LTR R15,R15 Was the UNSET successful?
 BNZ ERROR1 No, branch to error routine.
:
PLIST EQU EBW064
VARNAME DC C'GREETING',XL1'00'
SETVAL DC C'Hello, world!',XL1'15',XL1'00'
CHNGFLG DC F'1'

```

## CIFRC–Cipher Program Interface

An ECB-controlled program executes this general macro whenever a portion of a message to or from the IBM 3614 Consumer Transaction Facility needs to be enciphered or deciphered. Issuing the CIFRC macro causes the IBM-supplied cipher program BQKCIPH or BQKDES to be executed.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **TEXT**

This parameter is required and must be coded exactly as shown.

#### **PGM**

This parameter identifies the cipher program being requested. It must be either AET or DES. When this parameter is omitted a default is used. If support for only one cipher program is generated, that will be the default. If support for both cipher programs is generated, then the default is AET.

#### **AET**

Causes program BQKCIPH to be executed.

#### **DES**

Causes program BQKDES to be executed.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R1 must contain the address of a 3-word parameter list. The parameter list must contain the following information:

#### **Word    Contents**

- |   |                                                                                                                                                                                                                                                                                                                                                                                             |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | Request code: 0 indicates that the text is to be enciphered; any other value indicates that the text is to be deciphered.                                                                                                                                                                                                                                                                   |
| 1 | Address of an 8-byte key to use in the enciphering or deciphering. Any value is valid as a key.                                                                                                                                                                                                                                                                                             |
| 2 | Address of the text to be enciphered or deciphered. This is also the address at which the cipher program stores the enciphered or deciphered text. Storage protection is disabled when the text is stored, so the addressed main storage area may have any storage protection key. When AET is specified, four bytes of text are used. When DES is specified, eight bytes of text are used. |

Only the first 4 or 8 bytes at this address are enciphered or deciphered. To encipher or decipher more data, the CIFRC macro must be executed again.



## 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 input text pointed to by the third word (word 2) of the input parameter list is replaced by the enciphered or deciphered text.

## Programming Considerations

- This macro can be executed on any I-stream.
- For a discussion of 3614 programming considerations, including data enciphering, see the 3614 Programmer's Guide.

## Examples

None.

**E-type Program:**

```

graph LR
 Start(()) --> CINFC[CINFC]
 CINFC --> LIST[LIST]
 LIST --> Etype[E-type Program]
 LIST --> Ctype[C-type Program]
 Etype --> End(())
 Ctype --> End

```

**C-type Program:**

```

graph LR
 Start(()) --> Rintpt[R, intpt]
 Rintpt --> Wintpt[W, intpt]
 Wintpt --> Aintpt[A, intpt]
 Aintpt --> K[KEYB
KEYD
KEYE
KEY1
KEY2]
 K --> REGR14[REG=R14]
 K --> REGRx[REG=Rx]
 REGR14 --> BSSYES[BSS=YES]
 REGRx --> BSSNO[BSS=NO]
 BSSYES --> INDEX[INDEX]
 BSSNO --> INDEX
 INDEX --> TABLE[TABLE]
 TABLE --> End(())

```

A symbolic name can be assigned to the macro statement.

- ## 66 TPF V4R1 General Macros

For a C-type program, specifies the return interface address; leave storage protect key unchanged. All control program services will be by fast path. That is, the macro will always generate inline code for calls by the control program.

- A** For an E-type program, specifies the return interface location address; allow program to write in protected storage. This option is restricted and requires program authorization of KEY0 and RESTRICT.

For a C-type program, specifies the return interface address; leave storage protect key unchanged. All control program services will be by fast path. That is, the macro will always generate inline code for calls by the control program. In addition, the interface location address is returned.

- K** Initiate control program (CP) keypoint update. Specify one or more of the following:
- KEYB
  - KEYD
  - KEYE
  - KEY1
  - KEY2

### LIST

This option will generate a list of equates only and no executable code.

### *intpt*

This is the symbolic label of the desired interface point, as defined in the CINFC macro. The label is also called a CINFC tag.

For a current list of the CINFC tags refer to the source code for the CINFC macro.

- F** This is the optional fast path. It is valid with the R parameter only. The current protection key will not be changed and the macro will not request service via an SVC.

### BSS

This is the subsystem for which the macro will return the interface point. The defaults depend on whether the calling program is an E-type or part of the control program (C-type).

### **BSS=,**

When a null is the value of the parameter (BSS=,), the CE1DBI field (in the ECB) is assumed to contain the subsystem index. This is the default value for E-type programs. This type of invocation is processed by a service routine.

### **NO**

For an E-type program, the register specified in the REG parameter is assumed to have the subsystem index to be used to access the CINFC data. This type of invocation expands inline.

For a C-type program, the subsystem ordinal number must be supplied in REG prior to issuing the CINFC macro. The value is multiplied by 4 by CINFC to obtain the offset to the data.

### **YES**

For an E-type program, specifies that the BSS CINFC data will be accessed. This type of invocation expands inline.

For a C-type program, specifies that the value returned in REG will be for the basic subsystem. This is the default value for the control program.

## CINFC

### INDEX

The subsystem (ordinal) index value will be found in the register specified in the REG parameter. The user must multiply the value by 4 to obtain the offset to the data before calling CINFC.

This parameter is only meaningful in an MDBF environment.

### REG=R|R14

This is the register in which the interface address is to be placed. The default is R14, and R0 may not be specified. In addition, if the 'F' option is not specified, R14 must be specified for the 'REG' parameter.

### TABLE

This option is for CP use only and will generate a table of displacements within Keypoint X for all the keypointable keypoints. There will be two entries in the table for each keypointable keypoint: a) a displacement to the keypoint descriptor entry; and b) a displacement to the keypoint core address entry. Any CSECT which calls the CINFC TABLE also needs to call the CX0CK macro to define the symbols used to generate the table.

## Entry Requirements

- R9 must contain the address of the ECB being processed (for real-time segments only).
- For a CP call in which the parameter BSS=NO is coded, the general register specified by REG must contain the subsystem ordinal number.
- When BSS=INDEX is coded, the REG parameter must contain the correct subsystem index value.

## Return Conditions

- Control is returned to the next sequential instruction.
- For code R without the fast path option, the requested interface point address is in R14, and the current protection key is set to that of application core.
- For code R, with the fast path option, the requested interface point address is in R14 or the register specified in REG. The contents of R14 are unknown.
- For code W, the requested interface address is in R14, and the current protection key is set to zero.
- The original protection key is saved and can be restored when you issue the KEYRC macro with the OKEY=YES parameter coded.
- For code A, the requested interface location address is in R14 and the current protection key is set to zero.
- For code K, the update of the indicated keypoints has been scheduled but not completed.
- The contents of R15 are unknown, except when the (R) parameter is used with the fast path option and the REG parameter is R15.

## Programming Considerations

- The CINFC macro requires one, two, or three fullwords of storage. If the SVC is invoked (for codes R, W, A, and K), the macro expands to use one fullword. For code R with the fast path option, the macro expands to two fullwords, and when BSS=NO is included, the macro expands to three fullwords.
- Up to five control program keypoints can be updated with a single macro.

- Under code K, the following keypoints are in group 1: CTKB, CTKE, CTKD, CTK1, and CTK2. No keypoints are in group 2. The associated keypoint identifiers (to be specified as keypoint1, keypoint2, and so on) are KEYB, KEYE, KEYD, KEY1 and KEY2 (that is, to update keypoint D, specify KEYD, to update keypoint 1, specify KEY1, and so on.)
- For code R, if F is coded, macro CZ1CP will be invoked if it hasn't already been called. This is required for addressability to the CINFC table.
- The CINFC macro execution can result in a system error and a forced exit of the ECB if an address requested with the W option is not defined in this configuration. that is, the address is zero (0).
- The CINFC macro execution can result in a system error and a forced exit of the ECB if CINFC K is coded on an I-stream other than the main I-stream.
- For code K, the keypoint descriptors are altered according to the information contained in the parameter list so that keypoint updating will occur at the next opportunity.
- The CINFC K option of this macro can be executed only from the main I-stream. All other invocations of the macro can be executed from any I-stream.

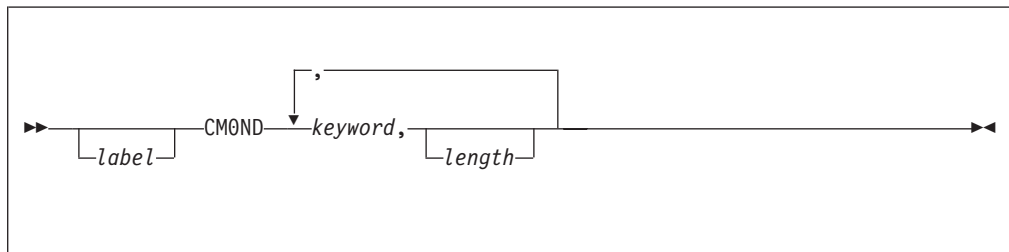
## Examples

None.

## CM0ND–Build Standardized Scan Tables

This general macro is used to build standardized keyword tables to be used directly by the application or indirectly by the CM0PR macro in scanning input messages.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *keyword*

This specifies the keyword to be scanned for. Keywords can be up to 8 characters in length.

#### *length*

This specifies the length of the minimum acceptable abbreviation of the keyword. If the length parameter is omitted, (for example, CM0ND START,,STOP,,), then it will default to the length of the keyword itself, (5 and 4 respectively, in the previous example) up to the maximum of 8 characters. If a comma appears at the end of a line without a following length field, the comma may be dropped.

### Entry Requirements

At least one keyword must be specified.

### Return Conditions

Control is returned to the next sequential instruction.

### Programming Considerations

- This is not an executable macro. The invocation of the CM0ND macro will lead to the inline expansion of the keywords into a standardized scan table.
- The CM0ND macro may be used on any I-stream.
- The keywords specified in the CM0ND table must not contain any of the following delimiters:
  - hyphen (-)
  - comma (,)
  - blank ( )
  - equal sign (=)
  - right parenthesis ())

### Examples

```
TABLE1 CM0ND DISPLAY,3,START,,DEFINE,3,LOCATE,3
```

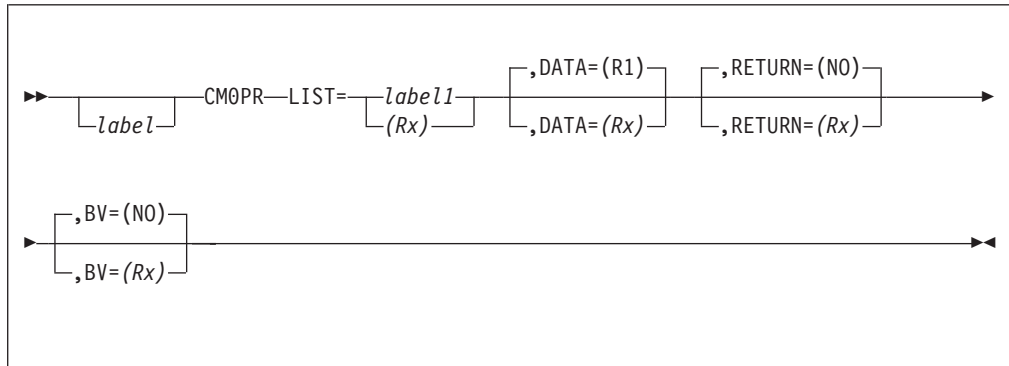
Will expand inline into the following table:

|        |    |              |                                    |
|--------|----|--------------|------------------------------------|
| TABLE1 | DS | 0H           |                                    |
|        | DC | AL1(3-1)     | (MINIMUM) COMMAND (EXECUTE) LENGTH |
|        | DC | AL1(4)       | BRANCH VECTOR                      |
|        | DC | CL8'DISPLAY' | COMMAND                            |
|        | DC | AL1(5-1)     | (MINIMUM) COMMAND (EXECUTE) LENGTH |
|        | DC | AL1(8)       | BRANCH VECTOR                      |
|        | DC | CL8'START'   | COMMAND                            |
|        | DC | AL1(3-1)     | (MINIMUM) COMMAND (EXECUTE) LENGTH |
|        | DC | AL1(12)      | BRANCH VECTOR                      |
|        | DC | CL8'DEFINE'  | COMMAND                            |
|        | DC | AL1(3-1)     | (MINIMUM) COMMAND (EXECUTE) LENGTH |
|        | DC | AL1(16)      | BRANCH VECTOR                      |
|        | DC | CL8'LOCATE'  | COMMAND                            |
|        | DC | XL1'FF'      | DENOTE END OF LIST                 |
|        | DS | 0H           |                                    |

## CM0PR–Scan Input Message for Keywords

Use this general macro, with the CM0ND macro to scan an input message for keyword commands. The CM0PR macro is passed a pointer to a table of expected keywords generated by CM0ND and a pointer to the input message to scan. The CM0PR macro scans the input message for the presence of one of the keywords. This macro skips insignificant characters: blank, comma (,), right parenthesis ()), hyphen (-), and equal sign (=).

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### LIST

This parameter specifies the pointer to the table of expected keywords generated by CM0ND. The valid inputs are:

*label1*

A symbolic address of the table.

**(Rx)**

A general register (other than R0 and R15) containing the address of the table.

**Note:** The LIST parameter has no defaults.

#### DATA

This parameter specifies the pointer to the input message being scanned. The valid inputs are:

**(Rx)**

A general register (other than R0, R14 and R15) containing the address of the input message being scanned.

**Note:** If the DATA parameter is omitted, R1 is assumed to be the pointer to the input message block.

#### RETURN

This specifies the address of a branch vector list to be given control upon completion of the macro. The first branch vector in the list is always given control if no keyword is found in the input message or end-of-message is detected. If the first keyword in the table is found in the input message, then control is given to the second branch vector. If the second keyword in the table is found in the input message, then control is given to the third branch vector, and so on. The valid inputs are:



**(Rx)**

This is a general register (other than R0, R14 and R15) containing the address of the branch vector list.

**(NO)**

The branch vector list is assumed to be coded at the next sequential instruction.

**Note:** If the RETURN parameter is omitted, the (NO) option is assumed.

**BV**

This specifies the register that is to contain the branch vector at the completion of the macro. The valid inputs are:

**(Rx)**

This is a general register that contains the branch vector resulting from the scan. If this parameter is coded, control will be either given to the NSI or to the address specified in the RETURN parameter. It will be the responsibility of the calling program to perform the appropriate indexing into a branch vector list according to the contents of Rx.

**(NO)**

No register is to be loaded with the branch vector.

**Note:** If the BV parameter is omitted, the (NO) option is assumed.

## Entry Requirements

- The LIST parameter must be coded to point to a keyword table generated by the CM0ND macro.
- A branch vector table containing N+1 branch instructions, where N is the number of entries in the keyword table. See the examples.

## Return Conditions

- If the BV parameter is coded with the (Rx) option, Rx will be loaded with the branch vector resulting from the scan. Control will then be given to either the NSI or to the address specified in the RETURN parameter. It will be the responsibility of the calling program to perform the appropriate indexing into a branch vector list according to the contents of Rx.
- If the BV parameter is coded with the (NO) option, or is omitted altogether, the macro will perform the appropriate indexing into a branch vector list according to the RETURN parameter.
- If the Return parameter is coded with the (Rx) option, control is transferred into the branch vector list addressed by (Rx). Otherwise, control is transferred into the branch vector list starting at the NSI.
- The register specified by the DATA parameter will point immediately past the recognized keyword or will remain unchanged if no keyword is found.
- The contents of R0, R14 and R15 are unknown.
- All other registers remain unchanged across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- This macro may be invoked by both C-type as well as E-type programs.
- This macro expands inline to a variable number of bytes, depending on the parameters coded.

## CM0PR

- Since the macro expansion is so large, the macro should only be coded once within any given program.
- If there is a need to invoke the macro multiple times to scan different keyword tables, CM0PR could be placed in a subroutine and the various macro parameters could be used to achieve the various desired scans. See the examples.

## Examples

In all the examples the following assumptions will be made:

- R3 has been initialized to point to the start of an input message.
- Two keyword tables exist:

```
TABLE1 CM0ND DISPLAY,3,START,,DEFINE,3,LOCATE,3
TABLE2 CM0ND STATUS,3,RESOURCES,3,BUFFER,3
```

Using the LIST parameter with the label option:

```
CM0PR LIST=TABLE1,DATA=(R3)
B ERRORTN BV for no match or EOM condition
B DISPRTN BV for DISplay keyword
B STRTRTN BV for START keyword
B DEFNRTN BV for DEFine keyword
B LOCTR TN BV for LOCate keyword
```

Using the LIST parameter with the (Rx) option:

```
LA R2, TABLE2 Get address of TABLE2
CM0PR LIST=(R2),DATA=(R3)
B ERRORTN BV for no match or EOM condition
B STATRTN BV for STATus keyword
B RESCRTN BV for RESources keyword
B BUFFRTN BV for BUFfer keyword
```

Using the CM0PR macro in a subroutine and the RETURN parameter:

```
LA R2, TABLE1 Get address of TABLE1
BAS R7, SCANRTN Go scan for keywords from TABLE1
B ERRORTN BV for no match or EOM condition
B DISPRTN BV for DISplay keyword
B STRTRTN BV for START keyword
B DEFNRTN BV for DEFine keyword
B LOCTR TN BV for LOCate keyword
.
.
LA R2, TABLE2 Get address of TABLE2
BAS R7, SCANRTN Go scan for keywords from TABLE2
B ERRORTN BV for no match or EOM condition
B STATRTN BV for STATus keyword
B RESCRTN BV for RESources keyword
B BUFFRTN BV for BUFfer keyword
.
.
SCANRTN DS 0H
CM0PR LIST=(R2),DATA=(R3),RETURN=(R7)
```

Calling the CM0PR macro in PGMY from PGMX and the BV parameter:

```
BEGIN NAME=PGMX
.
.
LA R2, TBLINDX2 Set table index for TABLE2
ENTRC PGMY Go scan for keywords from TABLE1
B **4(R1) Branch using the branch vector
B ERRORTN BV for no match or EOM condition
B STATRTN BV for STATus keyword
B RESCRTN BV for RESources keyword
```

```

B BUFFRTN BV for BUffer keyword
.
.

BEGIN NAME=PGMY
.
initialize R2 to point to correct keyword
table according to the supplied table index.
.
CM0PR LIST=(R2),DATA=(R3),BV=(R1)
BACKC Return to calling program

```

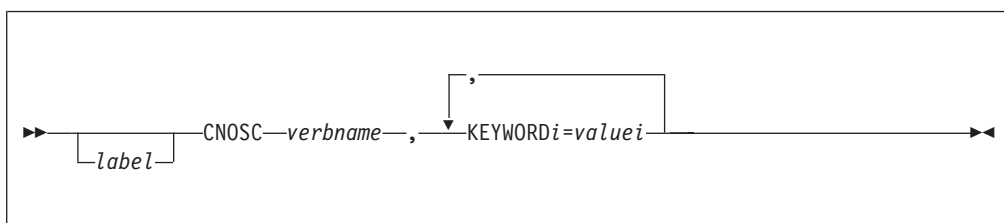
## CNOSC–TPF/APPC Change Number of Sessions Macro

Use this general macro to provide the communication interface between the control operator and the TPF application programs that are written to connect to remote nodes using the TPF/APPC session interface using LU 6.2 protocols.

**Note:** TPF does not support the DEFINE control operator verb defined in the LU 6.2 architecture. The CNOSC DISPLAY verb provides a subset of the parameters for the DISPLAY\_MODE verb defined by the LU 6.2 architecture.

This section contains a description of the CNOSC macro in its general form, followed by a separate section for each of the valid verbs.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *verbname*

Specifies the name of the TPF/APPC verb to be executed. The valid *verbnames* are shown in Table 2 on page 77.

#### **KEYWORD***i=valuei*

Specifies the one or more valid keyword parameters and the associated values.

Each macro call begins with CNOSC, followed by the positional parameter *verbname*, which specifies the verb to be executed. This is followed by one or more keyword parameters and values for those parameters. The values specify either a keyword option or a main storage location of a field.

The main storage location of a field can be specified as either the symbolic name of the field or as a register that points to the field. If you specify a register, the register name must be enclosed in parentheses and must be in the range of R1–R7.

Throughout the descriptions of the CNOSC macros, the phrase *source* LU refers to the TPF LU and the *target* LU refers to the remote LU.

Table 2 on page 77 is a summary of the supported verb names and their associated parameters. The table also contains:

- An indication of whether the parameter is passed to or returned from the TPF/APPC processing components
- A brief description of the parameter
- The equivalent parameter defined in the LU 6.2 architecture.

More detailed information for each parameter is provided in the individual verb descriptions later in this chapter.

Table 2. TPF/APPC Change Number of Session Verbs and Valid Keywords

| Verb Name | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                 | Architecture Equivalent           |
|-----------|-------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| CHANGE    | LUNAME            | Passed             | This specifies the name of the remote partner LU.                                                                                                                                                                           | LU_NAME                           |
|           | MODE              | Passed             | This specifies the mode name for which the session limit and polarities are to be changed.                                                                                                                                  | MODE_NAME                         |
|           | LIMIT             | Passed             | This specifies the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU for the specified mode name.                                                                             | LU_MODE_SESSION_LIMIT             |
|           | CONW              | Passed             | This specifies the minimum number of sessions of which the source LU is designated to be the contention winner.                                                                                                             | MIN_CONWINNERS_SOURCE             |
|           | CONL              | Passed             | This specifies the minimum number of sessions of which the target LU is designated to be the contention winner. (Or the minimum number of sessions of which the <i>source</i> LU is designated to be the contention loser.) | MIN_CONWINNERS_TARGET             |
|           | RESP              | Passed             | This specifies which LU is responsible for selecting and deactivating sessions as a result of a change that decreases the session limit.                                                                                    | RESPONSIBLE                       |
|           | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                     | RETURN_CODE                       |
| DISPLAY   | LUNAME            | Passed             | This specifies the name of the remote partner LU.                                                                                                                                                                           | FULLY_QUALIFIED_LU_NAME           |
|           | MODE              | Passed             | This specifies the mode name for which the session limit and polarities are to be displayed.                                                                                                                                | MODE_NAME                         |
|           | LIMIT             | Returned           | This specifies the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU for the specified mode name.                                                                             | LU_MODE_SESSION_LIMIT             |
|           | CONW              | Returned           | This specifies the minimum number of sessions of which the source LU is designated to be the contention winner.                                                                                                             | MIN_CONWINNERS                    |
|           | CONL              | Returned           | This specifies the minimum number of sessions of which the target LU is designated to be the contention winner. (Or the minimum number of sessions of which the <i>source</i> LU is designated to be the contention loser.) | MIN_CONLOSERS                     |
|           | RESP              | Returned           | This specifies which LU is responsible for selecting and deactivating sessions as a result of a change that decreases the session limit.                                                                                    | related to TERMINATION_COUNT      |
|           | DRAIN             | Returned           | This specifies whether the source LU or the target LU or both can drain its allocation requests.                                                                                                                            | DRAIN_LOCAL_LU<br>DRAIN_REMOTE_LU |

Table 2. TPF/APPC Change Number of Session Verbs and Valid Keywords (continued)

| Verb Name  | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                 | Architecture Equivalent |
|------------|-------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
|            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                     | RETURN_CODE             |
| INITIALIZE | LUNAME            | Passed             | This specifies the name of the remote partner LU.                                                                                                                                                                           | LU_NAME                 |
|            | MODE              | Passed             | This specifies the mode name for which the session limit and polarities are to be initialized.                                                                                                                              | MODE_NAME               |
|            | LIMIT             | Passed             | This specifies the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU for the specified mode name.                                                                             | LU_MODE_SESSION_LIMIT   |
|            | LOCAL             | Passed             | This specifies the local TPF/APPC LU.                                                                                                                                                                                       | Not applicable          |
|            | CONW              | Passed             | This specifies the minimum number of sessions of which the source LU is designated to be the contention winner.                                                                                                             | MIN_CONWINNERS_SOURCE   |
|            | CONL              | Passed             | This specifies the minimum number of sessions of which the target LU is designated to be the contention winner. (Or the minimum number of sessions of which the <i>source</i> LU is designated to be the contention loser.) | MIN_CONWINNERS_TARGET   |
|            | CDRM              | Passed             | This specifies the name of the remote cross-domain resource manager (CDRM) that owns the specified remote LU.                                                                                                               | Not applicable          |
|            | CP                | Passed             | This specifies the control point (CP) name of the adjacent APPN node where the specified remote LU resides.                                                                                                                 | Not applicable          |
|            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                     | RETURN_CODE             |
| RESET      | LUNAME            | Passed             | This specifies the name of the remote partner LU.                                                                                                                                                                           | LU_NAME                 |
|            | MODE              | Passed             | This specifies the mode name for which the session limit and polarities are to be reset.                                                                                                                                    | MODE_NAME               |
|            | RESP              | Passed             | This specifies which LU is responsible for selecting and deactivating sessions.                                                                                                                                             | RESPONSIBLE             |
|            | DRAIN             | Passed             | This specifies whether the source LU or the target LU or both can drain its allocation requests.                                                                                                                            | DRAIN_SOURCE            |
|            | FORCE             | Passed             | This specifies whether the source LU is to force the resetting of its session limit when certain error conditions occur that prevent successful exchange of the CNOS request and reply.                                     | FORCE                   |
|            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                     | RETURN_CODE             |

Table 2. TPF/APPC Change Number of Session Verbs and Valid Keywords (continued)

| Verb Name       | Keyword Parameter | Passed or Returned | Description                                                                                                   | Architecture Equivalent |
|-----------------|-------------------|--------------------|---------------------------------------------------------------------------------------------------------------|-------------------------|
| <b>Note:</b>    |                   |                    |                                                                                                               |                         |
| <b>Passed</b>   |                   |                    | Indicates that the value or option is passed from the transaction program to the verb processing component.   |                         |
| <b>Returned</b> |                   |                    | Indicates that the value or option is returned to the transaction program from the verb processing component. |                         |

## CNOSC

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- EBX030–EBX103 must be available for use. Also, parameters on CNOSC cannot point to a field within this range.
- Other entry requirements are dependent on the *verbname* specified. See the *verbname* section for specific information.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.
- The contents of EBX050–EBX103 are unknown.
- The TPF/APPC support always returns to the calling program with a protection key of 1.
- Other return conditions are dependent on the *verbname* specified. See the *verbname* section for specifics.
- The CNOSC macro returns a 6-byte return code. The first 2 bytes are the primary return code, and the next 4 bytes are the secondary return code. Table 3 contains a list of all the primary return codes and their meanings, and Table 4 on page 81 contains a list of all the secondary return codes and their meanings. Not all return codes are possible for every verb. The individual CNOSC macro descriptions document the valid return codes for that CNOSC macro. In addition, Table 5 on page 82 and Table 6 on page 83 show the correlation between the return codes and the verbs for which they are valid. The symbolic names of the return code values are defined in data macro ICN00S.

Table 3. CNOSC Primary Return Codes

| Symbolic Name           | Hex Value | Meaning                                                                                                                                                                                                                                                          |
|-------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CNOSRC_OK               | 0000      | The function completed successfully. Check the secondary return code for additional information.                                                                                                                                                                 |
| CNOSRC_PARAMETER_ERROR  | 0004      | An invalid parameter was detected. Check the secondary return code.                                                                                                                                                                                              |
| CNOSRC_ALLOC_ERROR      | 0008      | An allocation error occurred. Check the secondary return code.                                                                                                                                                                                                   |
| CNOSRC_RACE_REJECT      | 000C      | The CNOSC verb request failed because the source LU or target LU is currently processing another CNOS transaction for the same mode name. The source and target LU's CNOS parameters are not changed, and the other CNOS transaction is processed to completion. |
| CNOSRC_LIMIT_ZERO       | 0010      | The program attempted to change a session limit that was not initialized; that is, the session limit is 0.                                                                                                                                                       |
| CNOSRC_RESOURCE_FAILURE | 0014      | A resource failure occurred. Check the secondary return code for additional information.                                                                                                                                                                         |
| CNOSRC_LIMIT_CLOSED     | 0018      | The target LU will not currently allow the session limit for the specified mode name to be raised above zero. This condition is not necessarily permanent; retry the CNOS request later.                                                                         |



Table 3. CNOSC Primary Return Codes (continued)

| Symbolic Name            | Hex Value | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CNOSRC_LIMIT_NOT_ZERO    | 001C      | The program attempted to initialize a session limit that is already initialized; that is, the session limit is already greater than zero.                                                                                                                                                                                                                                                                                                                                                                                        |
| CNOSRC_INVALID_REQ       | 0020      | The request was invalid and failed. Check the secondary return code for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| CNOSRC_LOCAL_LU_MISMATCH | 0024      | The CNOSC INIT request was rejected because the local LU specified cannot be used. <ul style="list-style-type: none"> <li>– If the remote LU is in session with secondary LU (SLU) threads, the value on the LOCAL parameter must be a SLU thread.</li> <li>– If the remote LU is in session with a service LU, the value on the LOCAL parameter must be a service LU.</li> <li>– If the remote LU is in session with the generic TPF/APPC LU, the value on the LOCAL parameter must be the same generic TPF/APPC LU.</li> </ul> |

Table 4. CNOS Secondary Return Codes

| Symbolic Name             | Hex Value | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CNOSRC_AS_SPECIFIED       | 00000000  | The request is complete as specified.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| CNOSRC_AS_NEGOTIATED      | 00000004  | The request is complete but was negotiated by the target LU.                                                                                                                                                                                                                                                                                                                                                                                                                |
| CNOSRC_FORCED             | 00000008  | The RESET request could not be completed normally, so the force was done as specified.                                                                                                                                                                                                                                                                                                                                                                                      |
| CNOSRC_RETRY_LATER        | 0000000C  | The request failed. Retry the request later.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| CNOSRC_NO_RETRY           | 00000010  | The request failed. Do not retry the request.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| CNOSRC_INVALID_LU         | 00000014  | A parameter check was detected. The remote LU specified on the LUNAME parameter is invalid.                                                                                                                                                                                                                                                                                                                                                                                 |
| CNOSRC_INVALID_MODE       | 00000018  | A parameter check was detected. An invalid mode name was passed. This error occurs if: <ul style="list-style-type: none"> <li>– There is a request to change or reset a mode that was not previously initialized.</li> <li>– There is a request to initialize a parallel session mode name for a remote LU that has a single session.</li> <li>– There is a request to initialize a single session mode name for a remote LU that already has parallel sessions.</li> </ul> |
| CNOSRC_NON_ZERO_LIMIT_REQ | 0000001C  | A parameter check was detected. A session limit of zero was passed and a nonzero limit is required.                                                                                                                                                                                                                                                                                                                                                                         |
| CNOSRC_SNASVCMG_INVALID   | 00000020  | A parameter check was detected. The mode name SNASVCMG is not allowed for the CHANGE verb.                                                                                                                                                                                                                                                                                                                                                                                  |
| CNOSRC_SUM_ERROR          | 00000024  | A parameter check was detected. The sum of CONW and CONL exceeds the specified session limit.                                                                                                                                                                                                                                                                                                                                                                               |

## CNOSC

Table 4. CNOS Secondary Return Codes (continued)

| Symbolic Name             | Hex Value | Meaning                                                                                                                                                                                                                                                                                                         |
|---------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CNOSRC_ZERO_LIMITS_REQ    | 00000028  | A parameter check was detected. All session limits must be zero before SNASVCMG can be reset.                                                                                                                                                                                                                   |
| CNOSRC_ALREADY_RESET      | 0000002C  | A parameter check was detected. One cause for this is that a RESET request was issued to allow draining for the source LU; however, a previous RESET request disallowed draining for the source. Another cause is that a RESET MODE=ALL was issued but all mode names (other than SNASVCMG) were already reset. |
| CNOSRC_INVALID_STATE      | 00000030  | The specified request cannot be completed while the system is below CRAS state.                                                                                                                                                                                                                                 |
| CNOSRC_SINGLE_INVALID     | 00000038  | An invalid request was detected. The single session mode name is not allowed for the CHANGE verb.                                                                                                                                                                                                               |
| CNOSRC_LOCAL_LU_INVALID   | 0000003C  | A parameter check was detected. The local LU specified on the LOCAL parameter is invalid.                                                                                                                                                                                                                       |
| CNOSRC_INVALID_CDRM       | 00000040  | A parameter check. An invalid CDRM name was passed.                                                                                                                                                                                                                                                             |
| CNOSRC_CDRM_NOT_ACTIVE    | 00000044  | A parameter check. There is no CDRM-CDRM session with the specified remote CDRM.                                                                                                                                                                                                                                |
| CNOSRC_MAX_SESSION_EXCEED | 00000054  | A parameter check. The value specified for the session limit is greater than the maximum number of sessions permitted.                                                                                                                                                                                          |
| CNOSRC_HOST_VIOLATION     | 00000060  | A resource failure. Sessions exist for this LU on another processor.                                                                                                                                                                                                                                            |
| CNOSRC_LOCAL_NOT_ACTIVE   | 00000064  | A resource failure. The specified (or default) local LU is not active.                                                                                                                                                                                                                                          |
| CNOSRC_CP_NOT_VALID       | 00000068  | A parameter check. The CP name passed was not valid.                                                                                                                                                                                                                                                            |
| CNOSRC_CP_NOT_ACTIVE      | 0000006C  | A resource failure. No active ALS exists between the TPF system and specified adjacent CP.                                                                                                                                                                                                                      |

Table 5. Correlation of Primary Return Codes to CNOSC Verbs

| Primary Return Code (Value) | Verb   |         |            |       |
|-----------------------------|--------|---------|------------|-------|
|                             | CHANGE | DISPLAY | INITIALIZE | RESET |
| OK (0)                      | S      | X       | S          | S     |
| PARAMETER ERROR (4)         | S      | S       | S          | S     |
| ALLOC ERROR (8)             | S      |         | S          | S     |
| RACE REJECT (C)             | X      |         | X          | X     |
| LIMIT ZERO (10)             | X      |         |            |       |
| RESOURCE FAILURE (14)       | S      |         | S          | S     |
| LIMIT CLOSED (18)           |        |         | X          |       |
| LIMIT NOT ZERO (1C)         |        |         | X          |       |
| INVALID REQ (20)            | S      | X       | S          | S     |
| LOCAL LU MISMATCH (24)      |        |         | X          |       |

Table 5. Correlation of Primary Return Codes to CNOSC Verbs (continued)

| Primary Return Code (Value) | Verb                                    |         |            |       |
|-----------------------------|-----------------------------------------|---------|------------|-------|
|                             | CHANGE                                  | DISPLAY | INITIALIZE | RESET |
| <b>Note:</b>                |                                         |         |            |       |
| <b>X</b>                    | No secondary return code                |         |            |       |
| <b>S</b>                    | See secondary return code               |         |            |       |
| blank                       | Return code not possible for this verb. |         |            |       |

Table 6. Correlation of Secondary Return Codes to CNOSC Verbs

| Secondary Return Code (Value) | Verb   |         |            |       | Corresponding Primary Return Code Value |
|-------------------------------|--------|---------|------------|-------|-----------------------------------------|
|                               | CHANGE | DISPLAY | INITIALIZE | RESET |                                         |
| AS SPECIFIED (0)              | X      |         | X          | X     | (0)                                     |
| AS NEGOTIATED (4)             | X      |         | X          | X     | (0)                                     |
| FORCED (8)                    |        |         |            | X     | (0)                                     |
| RETRY LATER (C)               | X      |         | X          | X     | (8)                                     |
| NO RETRY (10)                 | X      |         | X          | X     | (8), (14), (18)                         |
| INVALID LU (14)               | X      | X       | X          | X     | (4)                                     |
| INVALID MODE (18)             | X      | X       | X          | X     | (4)                                     |
| NON ZERO LIMIT REQ (1C)       | X      |         | X          |       | (4)                                     |
| SNASVCMG INVALID (20)         | X      |         |            |       | (4)                                     |
| SUM ERROR (24)                | X      |         | X          |       | (4)                                     |
| ZERO LIMITS REQ (28)          |        |         |            | X     | (4)                                     |
| ALREADY RESET (2C)            |        |         |            | X     | (20)                                    |
| INVALID STATE (30)            | X      | X       | X          | X     | (20)                                    |
| SINGLE INVALID (38)           | X      | X       | X          | X     | (20)                                    |
| LOCAL LU INVALID (3C)         |        |         | X          |       | (4)                                     |
| INVALID CDRM (40)             |        |         | X          |       | (4)                                     |
| CDRM NOT ACTIVE (44)          |        |         | X          |       | (4)                                     |
| MAX SESSION EXCEED (54)       | X      |         | X          |       | (4)                                     |
| HOST VIOLATION (60)           |        |         | X          |       | (14)                                    |
| LOCAL NOT ACTIVE (64)         | X      |         | X          | X     | (14)                                    |
| CP NOT VALID (68)             |        |         | X          |       | (4)                                     |
| CP NOT ACTIVE (6C)            |        |         | X          |       | (4)                                     |

## Programming Considerations

- You can execute this macro on any I-stream.
- See the individual verb descriptions for programming considerations relating to the supported verbs.

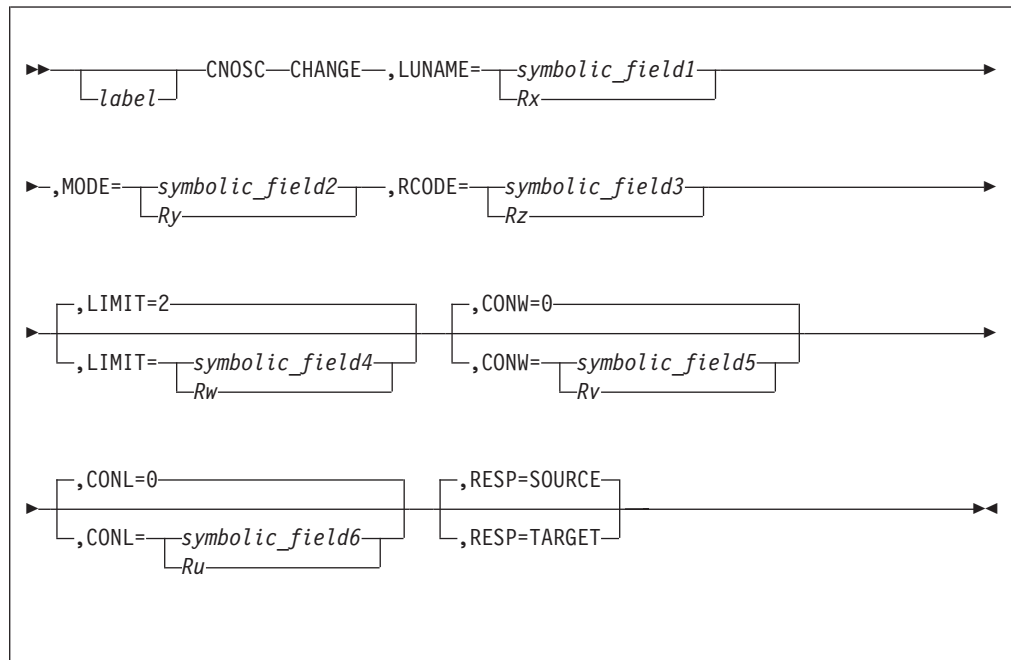
## Examples

See the individual verb sections for examples.

## CNOSC CHANGE

Use the CNOSC general macro with the CHANGE verb specified to change the LU 6.2 session limit and the contention-winner polarities for parallel session connections.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### CHANGE

Directs the CNOSC macro to perform the CHANGE verb function.

### LUNAME

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the partner (remote) LU to which the session limit and contention-winner polarity changes apply. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

### MODE

Specifies the symbolic name of a field or a register pointing to a field. This is an 8-byte field that contains the mode name for which the session limit and polarities are to be changed.

### LIMIT

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the session limit for the specified mode name. (The session limit is the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU.)

The specified limit must be greater than zero and greater than or equal to the sum of the numbers specified in the CONW and CONL parameters. If you do not specify LIMIT, the value defaults to 2.

**CONW**

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the minimum number of sessions for which the source LU is designated to be the contention winner.

The specified number must be zero or greater, and the sum of this number and the number specified for CONL cannot exceed the number specified with LIMIT. If you do not specify CONW, the value defaults to 0.

**CONL**

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the minimum number of sessions for which the source LU is designated to be the contention loser.

The specified number must be zero or greater, and the sum of this number and the number specified for CONW cannot exceed the number specified with LIMIT. If you do not specify CONL, the value defaults to 0.

**RESP**

Specifies the LU that is responsible for selecting and deactivating parallel sessions as a result of a change that decreases the session limit or the maximum number of contention-winners for the source or target LUs. The allowed values are:

**SOURCE**

Specifies that the source LU is responsible for selecting and deactivating sessions as a result of a change that decreases the session limit or the maximum number of contention-winners for the source or target LUs. The target LU cannot negotiate this argument. This is the default value.

**TARGET**

Specifies that the target LU is responsible for selecting and deactivating sessions as a result of a change that decreases the session limit or the maximum number of contention-winners for the source or target LUs. The target LU can negotiate this argument to make the source LU responsible.

**RCODE**

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions".

## Entry Requirements

See "Entry Requirements" on page 80 for the entry requirements relating to the CNOSC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section.
- See "Return Conditions" on page 80 for the return conditions relating to the CNOSC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the CHANGE verb. A complete list of return codes and their definitions can be found in Table 3 on page 80 and in Table 4 on page 81.

## CNOSC CHANGE

| Symbolic Name             | Primary Code | Secondary Code |
|---------------------------|--------------|----------------|
| CNOSRC_OK                 | 0000         |                |
| CNOSRC_AS_SPECIFIED       | ....         | 00000000       |
| CNOSRC_AS_NEGOTIATED      | ....         | 00000004       |
| CNOSRC_PARAMETER_ERROR    | 0004         |                |
| CNOSRC_INVALID_LU         | ....         | 00000014       |
| CNOSRC_INVALID_MODE       | ....         | 00000018       |
| CNOSRC_NON_ZERO_LIMIT_REQ | ....         | 0000001C       |
| CNOSRC_SNASVCMG_INVALID   | ....         | 00000020       |
| CNOSRC_SUM_ERROR          | ....         | 00000024       |
| CNOSRC_MAX_SESSION_EXCEED | ....         | 00000054       |
| CNOSRC_ALLOC_ERROR        | 0008         |                |
| CNOSRC_RETRY_LATER        | ....         | 0000000C       |
| CNOSRC_NO_RETRY           | ....         | 00000010       |
| CNOSRC_RACE_REJECT        | 000C         |                |
| CNOSRC_LIMIT_ZERO         | 0010         |                |
| CNOSRC_RESOURCE_FAILURE   | 0014         |                |
| CNOSRC_NO_RETRY           | ....         | 00000010       |
| CNOSRC_LOCAL_NOT_ACTIVE   | ....         | 00000064       |
| CNOSRC_INVALID_REQ        | 0020         |                |
| CNOSRC_INVALID_STATE      | ....         | 00000030       |
| CNOSRC_SINGLE_INVALID     | ....         | 00000038       |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- You cannot use the change function for the SNA-defined mode name, SNASVCMG, or for modes initialized as single sessions.
- The CHANGE verb starts a conversation with the LU name specified to exchange information in the specified mode name. The macro generates an ALLOCATE, a SEND to send the limit and polarity information to the remote LU, a RECEIVE to receive the reply. The remote LU then issues a DEALLOCATE. When you issue the CHANGE verb, the program gives up control while waiting for the reply from the remote LU.
- See “Programming Considerations” on page 83 for programming considerations relating to the CNOSC macro in general.

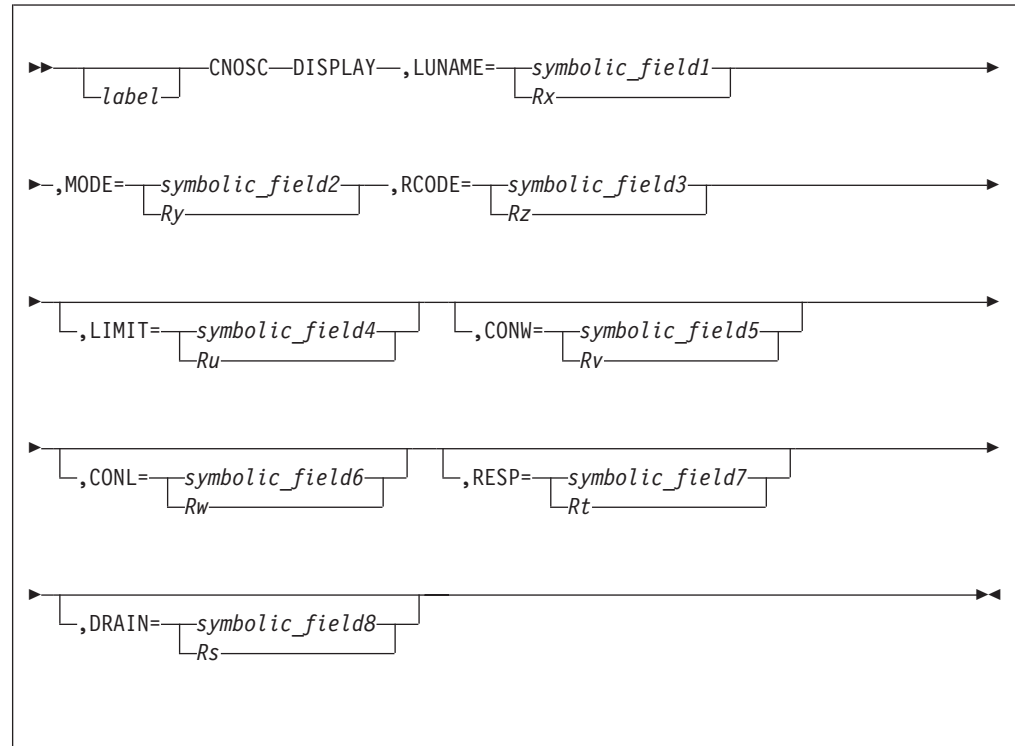
## Examples

```
SYMB100 CNOSC CHANGE, X
 LUNAME=EBX024, X
 MODE=EBX040, X
 LIMIT=EBW020, X
 CONW=EBW030, X
 CONL=EBW040, X
 RESP=TARGET, X
 RCODE=(R4)
```

## CNOSC DISPLAY

Use the CNOSC general macro with the DISPLAY verb specified to return the session limit, contention-winner polarities, drain status, and responsible LU status for a particular (LU,modename) pair.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### DISPLAY

Directs the CNOSC macro to perform the DISPLAY<sup>1</sup> verb function.

### LUNAME

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the partner (remote) LU to which the display applies. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

### MODE

Specifies the symbolic name of a field or a register pointing to a field. This is an 8-byte field that contains the mode name for which the session information is to be returned.

### LIMIT

Specifies the symbolic name of a field or a register pointing to a field. Upon return, this 2-byte field contains the session limit for the specified mode name.

1. This macro provides a subset of the parameters defined by the LU 6.2 architecture's DISPLAY\_MODE verb.

## CNOSC DISPLAY

(The session limit is the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU.)

### CONW

Specifies the symbolic name of a field or a register pointing to a field. Upon return, this 2-byte field contains the minimum number of sessions for which the local LU is designated to be the contention winner.

### CONL

Specifies the symbolic name of a field or a register pointing to a field. Upon return, this 2-byte field contains the minimum number of sessions for which the local LU is designated to be the contention loser.

### RESP

Specifies a symbolic name of a field or a register pointing to a field. Upon return, this 1-byte field contains a value indicating whether the local TPF LU or the remote LU is responsible for deactivating sessions for an outstanding CHANGE or RESET verb. The possible values are:

#### X'00'

No CHANGE or RESET verb pending.

#### X'01'

The remote LU is the responsible LU.

#### X'16'

The TPF LU is the responsible LU.

### DRAIN

Specifies a symbolic name of a field or a register pointing to a field. Upon return, this 1-byte field contains a value indicating whether the local TPF LU, the remote LU, both LUs, or neither LU can drain its allocation requests while there is a CNOSC RESET pending. The possible values are:

#### X'00'

No RESET verb pending, or neither LU can drain.

#### X'01'

The remote LU can drain its allocation requests.

#### X'16'

The TPF LU can drain its allocation requests.

#### X'17'

Both LUs can drain their allocation requests.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions".

## Entry Requirements

See "Entry Requirements" on page 80 for the entry requirements relating to the CNOSC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section.
- See "Return Conditions" on page 80 for the return conditions relating to the CNOSC macro in general.



## CNOSC DISPLAY

- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the DISPLAY verb. A complete list of return codes and their definitions can be found in Table 3 on page 80 and in Table 4 on page 81.

| Symbolic Name          | Primary Code | Secondary Code |
|------------------------|--------------|----------------|
| CNOSRC_OK              | 0000         |                |
| CNOSRC_PARAMETER_ERROR | 0004         |                |
| CNOSRC_INVALID_LU      | ....         | 00000014       |
| CNOSRC_INVALID_MODE    | ....         | 00000018       |
| CNOSRC_INVALID_REQ     | 0020         |                |
| CNOSRC_INVALID_STATE   | ....         | 00000030       |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- See “Programming Considerations” on page 83 for programming considerations relating to the CNOSC macro in general.

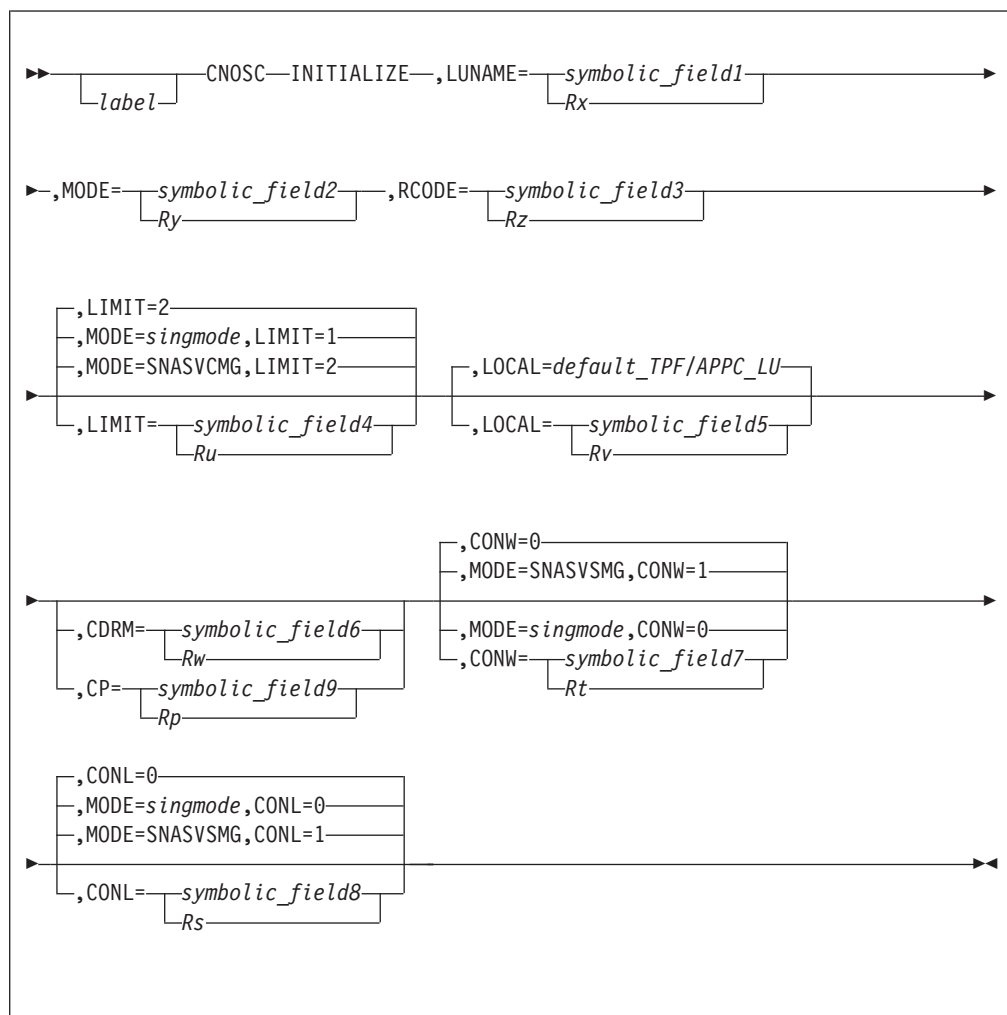
## Examples

```
SYMB100 CNOSC DISPLAY, X
 LUNAME=(R1), X
 MODE=(R3), X
 RESP=EBW020, X
 DRAIN=EBW021, X
 LIMIT=EBW030, X
 CONW=(R2), X
 CONL=(R5), X
 RCODE=(R4)
```

## CNOSC INITIALIZE

Use the CNOSC general macro with the INITIALIZE verb specified to initialize the LU 6.2 session limit and the contention-winner polarities for parallel or single session connections.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### INITIALIZE

Directs the CNOSC macro to perform the INITIALIZE verb function.

### LUNAME

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the partner (remote) LU to which the session limit and contention-winner polarity initialization applies. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

### MODE

Specifies the symbolic name of a field or a register pointing to a field. This is an

8-byte field that contains the mode name for which the session limit and polarities are to be initialized. You can specify a user-defined mode name or one of the following values:

### *singmode*

Specifies the mode name for single sessions as defined by the SINGMODE parameter of the SNAKEY macro. See *TPF ACF/SNA Network Generation* for more information about the SNAKEY macro.

### **SNASVCMG**

Specifies the SNA-defined mode name used to exchange CNOS verbs between the source and target LUs connected by parallel sessions.

### **LIMIT**

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the session limit for the specified mode name. (The session limit is the maximum number of sessions to be allowed between the source (TPF) LU and the target (remote) LU.)

The specified limit must be greater than zero and greater than or equal to the sum of the numbers specified in the CONW and CONL parameters. If you do not specify LIMIT, the value defaults to 2.

If you specify MODE=SNASVCMG, do not specify LIMIT; the value is set to 2.

If you specify MODE=*singmode*, do not specify LIMIT; the value is set to 1.

### **LOCAL**

Specifies the symbolic name of a field or a register pointing to a field. This is an 8-byte field that contains the network name of the local TPF/APPC LU, which is left-justified and padded with blanks.

If you do not specify LOCAL, the value is the default local TPF/APPC LU. The default local TPF/APPC LU is defined using the MSGRTA macro. See *TPF System Generation* for more information about the MSGRTA macro.

### **CDRM**

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the remote cross-domain resource manager (CDRM) that owns the specified remote LU. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the CDRM name is unqualified. The second 8 bytes contain the left-justified CDRM name, which is padded with blanks.

### **CONW**

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the minimum number of sessions for which the source LU is designated to be the contention winner.

The specified number must be zero or greater and the sum of this number and the number specified for CONL cannot exceed the number specified with LIMIT. If you do not specify CONW, the value defaults to 0.

If you specify MODE=SNASVCMG, do not specify CONW; the value is set to 1.

If you specify MODE=*singmode*, do not specify CONW; the value is set to 0.

### **CONL**

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the minimum number of sessions for which the source LU is designated to be the contention loser.

## CNOSC INITIALIZE

The specified number must be zero or greater and the sum of this number and the number specified for CONW cannot exceed the number specified with LIMIT. If you do not specify CONL, the value defaults to 0.

If you specify MODE=SNASVCMG, do not specify CONL; the value is set to 1.

If you specify MODE=*singmode.*, do not specify CONL; the value is set to 0.

### CP

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the control point (CP) name of the adjacent APPN node where the specified remote LU resides. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the CP name is unqualified. The second 8 bytes contain the left-justified CP name, which is padded with blanks.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions".

## Entry Requirements

See "Entry Requirements" on page 80 for the entry requirements relating to the CNOSC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section.
- See "Return Conditions" on page 80 for the return conditions relating to the CNOSC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the INITIALIZE verb. A complete list of return codes and their definitions can be found in Table 3 on page 80 and in Table 4 on page 81.

| Symbolic Name             | Primary Code | Secondary Code |
|---------------------------|--------------|----------------|
| CNOSRC_OK                 | 0000         |                |
| CNOSRC_AS_SPECIFIED       | ....         | 00000000       |
| CNOSRC_AS_NEGOTIATED      | ....         | 00000004       |
| CNOSRC_PARAMETER_ERROR    | 0004         |                |
| CNOSRC_INVALID_LU         | ....         | 00000014       |
| CNOSRC_INVALID_MODE       | ....         | 00000018       |
| CNOSRC_NON_ZERO_LIMIT_REQ | ....         | 0000001C       |
| CNOSRC_SUM_ERROR          | ....         | 00000024       |
| CNOSRC_LOCAL_LU_INVALID   | ....         | 0000003C       |
| CNOSRC_INVALID_CDRM       | ....         | 00000040       |
| CNOSRC_CDRM_NOT_ACTIVE    | ....         | 00000044       |
| CNOSRC_MAX_SESSION_EXCEED | ....         | 00000054       |
| CNOSRC_CP_NOT_VALID       | ....         | 00000068       |
| CNOSRC_CP_NOT_ACTIVE      | ....         | 0000006C       |

| Symbolic Name            | Primary Code | Secondary Code |
|--------------------------|--------------|----------------|
| CNOSRC_ALLOC_ERROR       | 0008         |                |
| CNOSRC_RETRY_LATER       | ....         | 0000000C       |
| CNOSRC_NO_RETRY          | ....         | 00000010       |
| CNOSRC_RACE_REJECT       | 000C         |                |
| CNOSRC_RESOURCE_FAILURE  | 0014         |                |
| CNOSRC_NO_RETRY          | ....         | 00000010       |
| CNOSRC_HOST_VIOLATION    | ....         | 00000060       |
| CNOSRC_LOCAL_NOT_ACTIVE  | ....         | 00000064       |
| CNOSRC_LIMIT_CLOSED      | 0018         |                |
| CNOSRC_LIMIT_NOT_ZERO    | 001C         |                |
| CNOSRC_INVALID_REQ       | 0020         |                |
| CNOSRC_INVALID_STATE     | ....         | 00000030       |
| CNOSRC_LOCAL_LU_MISMATCH | 0024         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The INITIALIZE verb starts a conversation with the LU name specified to exchange information in the specified mode name. The macro generates an ALLOCATE, a SEND to send the limit and polarity information to the remote LU, a RECEIVE to receive the reply. The remote LU then issues a DEALLOCATE. When you issue the INITIALIZE verb, the program gives up control while waiting for the reply from the remote LU.

**Note:** This does not apply to single session mode names or the SNASVCMG mode name.

- If you specify MODE=SNASVCMG, the value for LIMIT is 2, and the value for CONW and CONL is 1. In this case, if you code LIMIT, CONW, and CONL they are ignored.
- If you specify a single session mode name, the value for LIMIT is 1, and the value for CONW and CONL is 0. In this case, if you code LIMIT, CONW, and CONL they are ignored.
- If there are already parallel session modes initialized with the specified remote LU, you cannot initialize the single session mode.
- If there is a single session mode initialized with the specified remote LU, you cannot initialize parallel sessions.
- If you are not using secondary LU (SLU) threads, use the LOCAL parameter only when the first mode is initialized for a particular remote LU. For subsequent initializations, the same local TPF/APPC LU is used regardless of the value specified for this parameter. After all modes are reset for this remote LU, a different local TPF/APPC LU can be used.
- If you are using SLU threads, you must code LOCAL for each initialize, specifying different thread names each time.
- TPF keeps session limit and local LU information until a CNOSC RESET is done. This allows sessions to be recovered after a system or link failure without another CNOSC INITIALIZE.

## CNOSC INITIALIZE

- If you do not specify the CDRM keyword or the CP keyword, how the TPF system starts sessions with the specified remote LU is determined by the following:
  - Whether the TPF system is connected to the network as a PU 5 node, PU 2.1 node, or both.
  - How sessions with the specified remote LU were activated previously.

See *TPF ACF/SNA Data Communications Reference* for more information about how the TPF system selects the path to use when starting a new LU-LU session.

- See “Programming Considerations” on page 83 for programming considerations relating to the CNOSC macro in general.

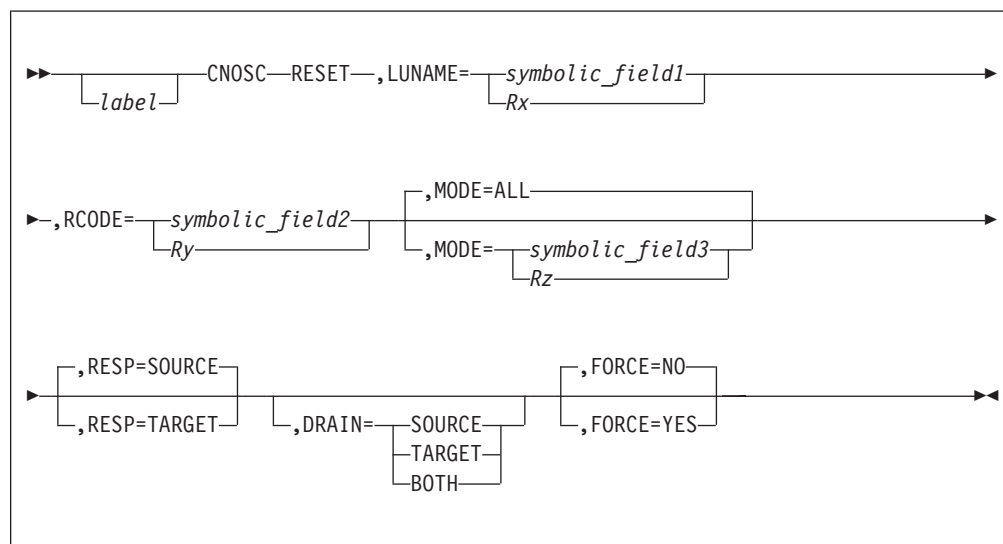
## Examples

```
SYMB100 CNOSC INITIALIZE, X
 LUNAME=EBX024, X
 MODE=EBW080, X
 LIMIT=EBW020, X
 CONW=(R4), X
 CONL=(R3), X
 RCODE=EBX000
```

## CNOSC RESET

Use the CNOSC general macro with the RESET verb specified to reset the LU 6.2 session limit for parallel or single session connections to 0.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### RESET

Directs the CNOSC macro to perform the RESET verb function.

#### LUNAME

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the partner (remote) LU to which the session limit and contention-winner polarity reset applies. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

#### MODE

Specifies the symbolic name of a field or a register pointing to a field. This is an 8-byte field that contains the mode name for which the session limit and polarities are to be reset.

You can specify a user-defined mode name or one of the following values:

##### ALL

Specifies that the session limit for all mode names that apply to the target LU are reset to 0, except the SNA-defined mode SNASVCMG, which remains unchanged. This is the default value.

##### *singmode*

Specifies the mode name for single sessions as defined by the SINGMODE parameter of the SNAKEY macro. See *TPF ACF/SNA Network Generation* for more information about the SNAKEY macro.

##### SNASVCMG

Specifies the SNA-defined mode name used to exchange CNOS verbs between the source and target LUs connected by parallel sessions.

## CNOSC RESET

### RESP

Specifies which LU is responsible for deactivating the session as a result of resetting the session limit for parallel sessions. This parameter is not applicable to single session connections or the SNASVCMG session. The allowed values are:

#### SOURCE

Specifies that the source LU is responsible for deactivating sessions as a result of resetting the session limit. The target LU cannot negotiate this argument. This is the default value.

#### TARGET

Specifies that the target LU is responsible for deactivating sessions as a result of resetting the session limit. The target LU can negotiate this argument to make the source LU responsible.

### DRAIN

Specifies which LU (or both) can drain its allocation requests. If you do not code this parameter, no draining is allowed for either LU, and all outstanding and subsequent allocation requests will be rejected. This parameter is not valid for SNASVCMG connections. The allowed values are:

#### SOURCE

Specifies that the source LU can drain its allocation requests. The target LU cannot negotiate this argument. The source LU continues to allocate conversations to the sessions until no requests are waiting for allocation, at which time its draining ends. Allocation requests subsequent to the completion of draining are rejected.

#### TARGET

Specifies that the target LU can drain its allocation requests. The target LU can reject this argument; in this case no draining takes place.

The target LU continues to allocate conversations to the sessions until no requests are waiting for allocation, at which time its draining ends. Allocation requests subsequent to the completion of draining are rejected. This value is not valid for single session connections.

#### BOTH

Specifies that both the source and target LUs can drain their allocation requests. This value is not valid for single session connections.

### FORCE

Specifies whether the source LU should force the resetting of its session limit upon the unsuccessful completion of the CNOS request or reply exchange. This parameter does not apply to SNASVCMG or single session connections. The allowed values are:

**Y** Specifies that a reset should be forced. In this case, RESP defaults to SOURCE and no draining by either source or target LU takes place.

**N** Specifies that a reset should not be forced. The session limit is reset only upon successful completion of the CNOS request or reply exchange. This value is the default.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions" on page 97.



## Entry Requirements

See “Entry Requirements” on page 80 for the entry requirements relating to the CNOSC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section.
- See “Return Conditions” on page 80 for the return conditions relating to the CNOSC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the RESET verb. A complete list of return codes and their definitions can be found in Table 3 on page 80 and in Table 4 on page 81.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| CNOSRC_OK               | 0000         |                |
| CNOSRC_AS_SPECIFIED     | ....         | 00000000       |
| CNOSRC_AS_NEGOTIATED    | ....         | 00000004       |
| CNOSRC_FORCED           | ....         | 00000008       |
| CNOSRC_PARAMETER_ERROR  | 0004         |                |
| CNOSRC_INVALID_LU       | ....         | 00000014       |
| CNOSRC_INVALID_MODE     | ....         | 00000018       |
| CNOSRC_ZERO_LIMITS_REQ  | ....         | 00000028       |
| CNOSRC_ALLOC_ERROR      | 0008         |                |
| CNOSRC_RETRY_LATER      | ....         | 0000000C       |
| CNOSRC_NO_RETRY         | ....         | 00000010       |
| CNOSRC_RACE_REJECT      | 000C         |                |
| CNOSRC_RESOURCE_FAILURE | 0014         |                |
| CNOSRC_NO_RETRY         | ....         | 00000010       |
| CNOSRC_LOCAL_NOT_ACTIVE | ....         | 00000064       |
| CNOSRC_INVALID_REQ      | 0020         |                |
| CNOSRC_ALREADY_RESET    | ....         | 0000002C       |
| CNOSRC_INVALID_STATE    | ....         | 00000030       |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The RESET verb starts a conversation with the LU name specified to exchange information in the specified mode name. The macro generates an ALLOCATE, a SEND to send the limit and polarity information to the remote LU, a RECEIVE to receive the reply. The remote LU then issues a DEALLOCATE. When you issue the RESET verb, the program gives up control while waiting for the reply from the remote LU.

**Note:** This does not apply to single session mode names or the SNASVCMG mode name.

## CNOSC RESET

- See “Programming Considerations” on page 83 for programming considerations relating to the CNOSC macro in general.

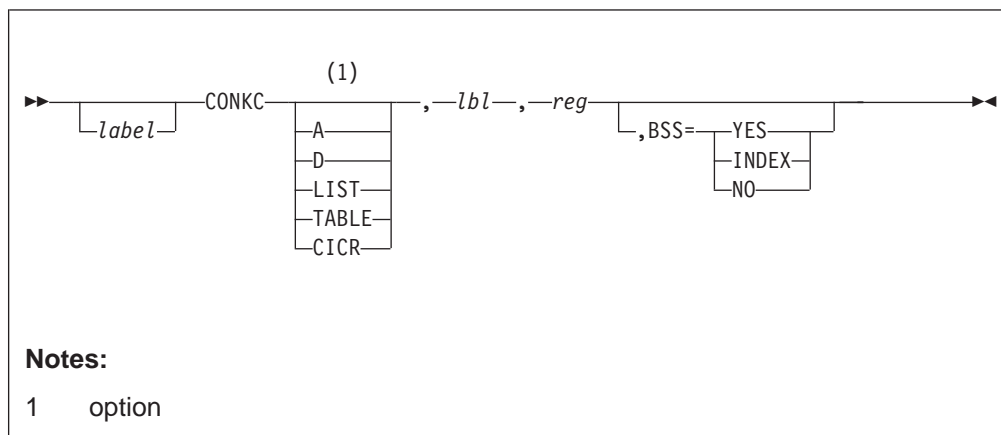
## Examples

```
SYMB100 CNOSC RESET, X
 LUNAME=(R1), X
 MODE=ALL, X
 RESP=SOURCE, X
 DRAIN=BOTH, X
 FORCE=Y, X
 RCODE=(R4)
```

## CONKC—Configuration Constants

This general macro provides access to a table of system configuration-dependent constants. This allows the user to alter the program function, depending on the configuration, without having to reassemble programs for every system configuration change.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *option*

Specify one of the following:

**A** The address of the requested data will be returned.

**D** The requested data will be returned.

#### **LIST**

List equates only. No address or data is returned.

#### **TABLE**

Generate the CONKC table. For use by the system to generate the table used to hold the actual configuration values.

#### **CICR**

Used by the system to generate the actual system SVC support code to access the configuration values.

*lbl* Label of address or data requested in the following format:

- 1st character is @
- 2nd and 3rd characters are numeric values from 00–15

Where:

**00** indicates the field is a fullword

**01** indicates the field is a halfword

**02** indicates the field is a byte

#### **03–15**

indicates unique table references (refer to CONKC Macro)

- 4th through 8th characters specify the remainder of the label.

## CONKC

*reg*

Register in which the requested address or data will be returned. R8, R9, R11, R12, and R13 may not be used by an E-type program. R0 cannot be used by a CP program.

### BSS

Is used by a CP segment to denote which subsystem's CONKC value is to be retrieved. This parameter is required when CONKC is coded by a CP segment. CONKC will not default to the BSS.

### YES

Access the BSS table.

### INDEX

The specified register *reg* contains the subsystem index times four of the target subsystem table.

### NO

The specified register *reg* contains the subsystem index of the target subsystem table.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction.
- The general register specified *reg* contains the data or the address of the data requested.

The contents of R14 and R15 are unknown, unless they were specified as input parameters (*reg*). The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- The information returned to the user in the requested register may be either a fullword, halfword or 1 byte in length. Information of less than a fullword in length will be right adjusted in the register, and the high-order bytes will be set to zeros. Registers R8, R9, R11, R12, R13 may not be used by an E-type program. R0 can not be used by a CP program.

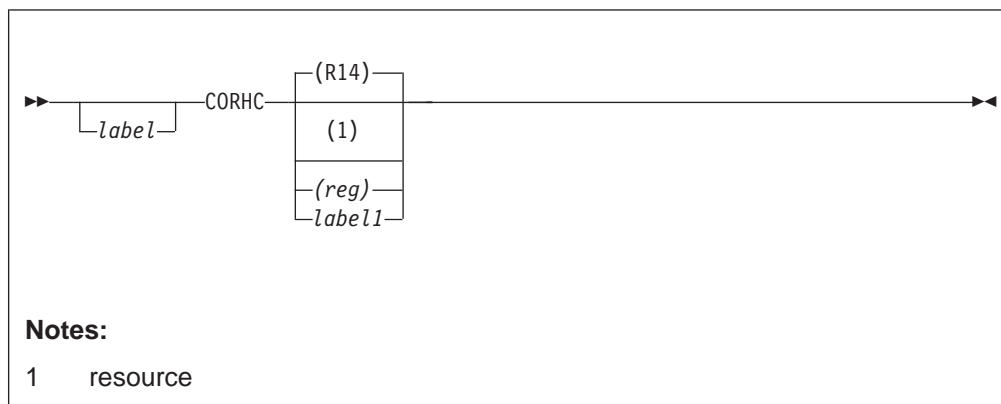
## Examples

None.

## CORHC—Define and Hold Resource

This general macro is used to define to the control program a shared resource and to control access to the resource among ECBs. The macro returns control to the calling ECB only when the shared resource is available for use by the ECB. This macro is used with the CORUC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*resource*

Is the 32-bit resource identification to be held, coded in either of the following formats:

*(reg)*

A register which contains the 32-bit resource identification to be held.

*label1*

A symbolic label that is a 4-byte field containing the 32-bit resource identification to be held.

This parameter is optional. If it is not coded, R14 must contain the 32-bit resource identification to be held.

### 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 R14 and R15 are unknown. The contents of R0-R7 are preserved across this macro call.

### Programming Considerations

- This macro can be executed on any I-stream.
- When finished with the resource, the ECB must issue an CORUC macro. If the ECB exits holding a resource, a system error is issued and the resource is freed.
- An error occurs if the ECB attempts to hold a resource which it is already holding.

## **CORHC**

- The CORHC macro is similar to coding an ENQC macro with parameters of TIMEOUT=0 and QUAL=S. This alternative to the ENQC macro is supplied to provide compatibility for some existing applications.

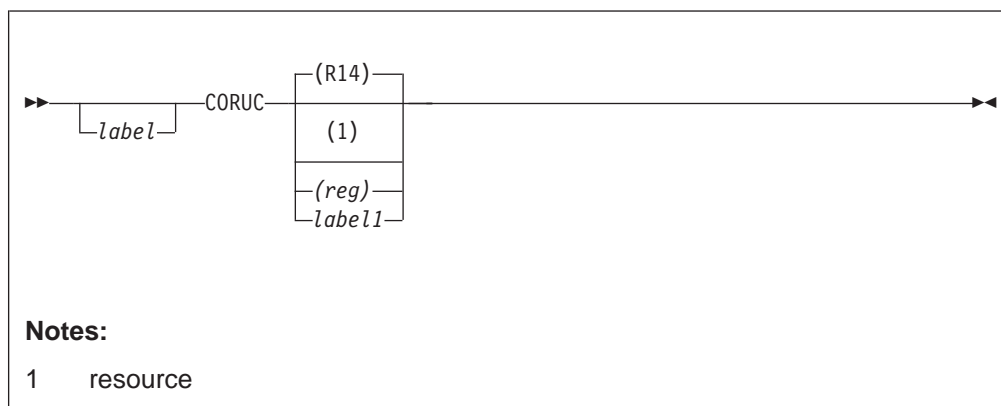
## **Examples**

None.

## CORUC–Unhold Resource

This general macro is used to notify the control program that an ECB no longer requires exclusive control of a shared resource. It is used in conjunction with the CORHC macro (See “CORHC–Define and Hold Resource” on page 101).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *resource*

Is the 32-bit resource identification to be unheld, coded in either of the following formats:

#### *(reg)*

A register which contains the 32-bit resource identification to be unheld.

#### *label1*

A symbolic label which is a 4-byte field containing the 32-bit resource identification to be unheld.

This parameter is optional. If it is not coded, R14 must contain the 32-bit resource identification to be unheld.

## 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 R14 and R15 are unknown. The contents of R0-R7 are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- An error occurs if the ECB attempts to unhold a resource which it is not holding.
- The CORUC macros is similar to coding a DEQC macro with the parameter QUAL=S. This alternative to the DEQC macro is supplied in order to provide compatibility for some existing applications. (See “DEQC–Dequeue from Resource” on page 150.)

**CORUC**

## **Examples**

None.

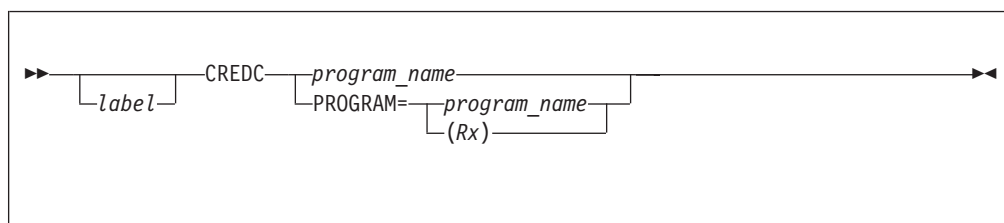


## CREDC—Create a Deferred Entry

This general macro creates an independent ECB for deferred processing. The ECB is created on the same I-stream, in the same subsystem, and subsystem user as the creating ECB.

A variable number of bytes (0 to 104) is passed to the created ECB work area. The control program moves the parameters into an interim block of available storage and adds this block to the deferred processing CPU loop list. Operational Program Zero (OPZERO) initializes an ECB with the parameters in the work area, releases the interim block, and executes an Enter with no return (ENTNC) to the specified program.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*program\_name*

The symbolic program name of the program that is to be activated with the created ECB must be specified. Either this positional parameter or the following keyword parameter can be used.

#### PROGRAM

The symbolic program name or a register containing the address of the program that is to be activated with the created ECB must be specified. This method generates constant data which used at execution time to determine the PAT displacement. This method has a slightly longer path length than the positional parameter method.

*program\_name*

The name of the program that is to be entered.

*(Rx)*

A register (R0–R7) that contains the address of the program name.

### Entry Requirements

- R14 must contain the number of bytes of parameters to be passed to the created ECB work area.
- R15 must contain the address of the start of the parameters which are to be passed.

### Return Conditions

- Control is returned to the instruction following the macro expansion.
- The contents of R14 and R15 and the condition code are unknown. The remaining operational program registers are saved during the execution of the CREDC macro.

## CREDC

### Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB issuing the CREDC macro.
- No linkage is provided between the created ECB and the active program segment using this macro.
- Up to 104 bytes of parameters can be passed to the created ECB.
- The ECB issuing the CREDC macro may be forced into a WAIT if there is insufficient storage available to buffer the parameters. When adequate working storage is available, the CREDC macro is executed and a return is made to the instruction after the macro expansion.
- Limit use of this macro to prevent storage depletion.
- The program (indicated by *program\_name*) must either be allocated in the PAT or be loaded online prior to execution. If the program is not either allocated or loaded, a system error results. See the *TPF System Installation Support Reference* for information on allocating programs and on the E-type loader.
- The macro trace for this macro contains the name of the macro that called it as well as the normal trace data.

### Examples

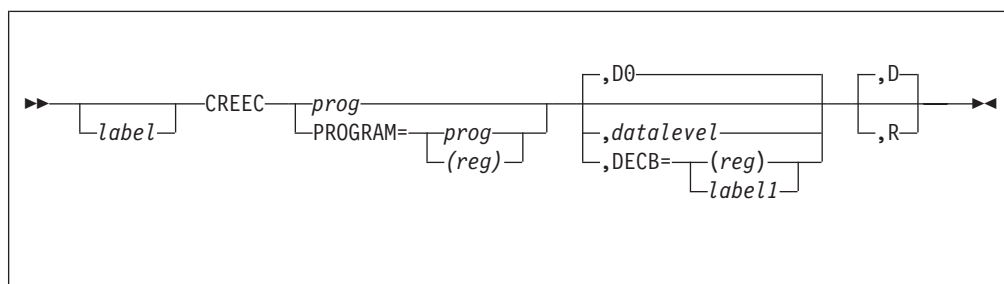
None.

## CREEC—Create a New ECB with Attached Core Blocks

This general macro creates an independent entry control block (ECB) for immediate or deferred processing. The ECB is created on the same I-stream, in the same subsystem, and subsystem user as the creating ECB.

A storage block on data level 0 and an optional variable-sized (0–104 bytes) parameter list are passed to the created ECB. The control program moves the specified core block reference word (CBRW) and the parameters into an interim block of available storage and adds the block to either the deferred processing list or the postinterrupt (ready) list. Operational Program Zero (OPZERO) initializes an ECB with the CBRW on data level 0, initializes the parameters in the work area, releases the interim block, and starts the specified program with an Enter with No Return (ENTNC).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *prog*

The name of the program that is to be activated with the created ECB. The name must be four character starting with an alphabetic character (A through Z). This method generates a V-con which is resolved at link edit time into a Program Allocation Table (PAT) displacement. This is the preferred method for specifying the program name.

#### **PROGRAM**

The name of the program can be provided using the PROGRAM parameter.

Either the positional parameter 'prog' or the keyword parameter 'PROGRAM' is required. This method generates constant data which will be used at execution time to determine the Program Allocation Table (PAT) displacement. This method has a longer path length than the one described above.

#### *prog*

The name of the program that is to be activated with the created ECB.

#### *(reg)*

A register (R0–R7) that contains the address of the program name.

#### *datalevel*

The data level of the block to be passed to the created ECB. The valid values are D0 through DF.

When the new ECB is dispatched, the block from the data level of the issuing ECB will be available to the new ECB on its data level 0.

## CREEC

When control is returned to the issuing ECB, the storage block previously attached at the level is detached and is no longer available for use by the issuing ECB.

**DECB**=(*reg*)/*label*1

The label or general register (R1–R7) containing the address of the data event control block (DECB) that contains the block to be passed to the created ECB.

When the new ECB is dispatched, the block from the DECB of the issuing ECB will be available to the new ECB on data level 0.

When control is returned to the issuing ECB, the storage block previously attached at the DECB is detached and is no longer available for use by the issuing ECB.

- R** Specifies that the created ECB is to be dispatched with a high priority. If you specify this parameter, the ECB is placed on the post interrupt (ready) list.
- D** Specifies that the created ECB is to be dispatched with a low priority. If you specify this parameter, the ECB is placed on the deferred process list.

## Entry Requirements

- The ECB reference register (R9) contains the address of the ECB issuing the CREEC macro.
- R14 must contain the number of bytes of parameters to be passed to the created ECB work area. If there are no parameters to pass, R14 must be set to zero.
- R15 must contain the address of the start of the parameters which are to be passed.
- There must be a storage block on the ECB data level or DECB specified. If a level or DECB is not specified, there must be a storage block on ECB data level 0.

## 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 in the ECB that issued the CREEC macro is set to indicate 'no block held'.

## Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB issuing the CREEC macro.
- No linkage is provided between the created ECB and the active program segment using this macro.
- There may be up to 104 bytes of parameters passed to the created ECB's work area.
- The ECB issuing the CREEC macro may be forced into a WAITC if there is insufficient storage available to buffer the parameters. When adequate storage is available, the CREEC macro is executed and control is returned to the next sequential instruction (NSI).
- **The use of this macro should be restricted in order to prevent depletion of storage.**

- Using the PROGRAM parameter sacrifices some performance. Consequently its use should be monitored.
- The program to be activated must have been allocated via the system allocator (refer to *TPF System Installation Support Reference*).
- When activated, the specified program receives control in its allocated addressing mode. The condition code and the contents of registers R0-R7, R14, and R15 are unpredictable.
- System errors can occur for the following reasons:
  - No core block is held at the specified ECB data level or DECB
  - More than 104 bytes of parameters are passed
  - The specified program has not been allocated.
- The core block specified by the ECB data level or DECB is released when this macro is processed. Once the core block is released, references to it cause errors. For example, an error results when the parameter list is put in the core block.
- This macro can be used only within a subsystem to pass a storage block to another program. Whenever possible, a storage block should be passed to another program using a ROUTC macro. This macro has been supplied to provide compatibility for some existing applications.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.
- 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.
- 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 and the entry is exited.

## Examples

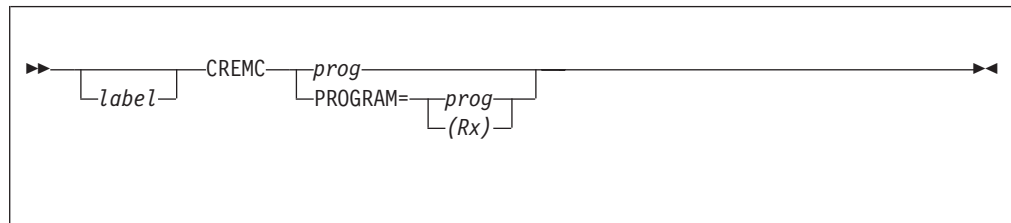
None.

## CREMC—Create a New ECB for Immediate Entry

This general macro creates an independent entry control block (ECB) for immediate processing by the requested program. The ECB is created on the same I-stream, in the same subsystem, and subsystem user as the creating ECB.

A variable number of bytes (0–104) is passed to the created ECB work area. The control program moves this data into an interim block of available storage and adds this block to the Ready List. Operational Program Zero (OPZERO) will initialize an ECB with the data in the work area, release the interim block, and execute an Enter with No Return (ENTNC) to the specified program.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be activated with the created ECB. This method generates a V-con which is resolved at link edit time into a Program Allocation Table (PAT) displacement. This is the preferred method for specifying the program name.

#### PROGRAM

The name of the program can alternately be provided via the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the Program Allocation Table (PAT) displacement. This method has a longer path length than the one described above.

*prog*

The name of the program that is to be activated with the created ECB.

*(Rx)*

A register (R0–R7) that contains the address of the program name.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must contain the number of bytes of data to be passed to the created ECB work area.
- R15 must contain the address of the start of the data which is to be passed.

### Return Conditions

- Control is returned to the instruction following the macro expansion.
- 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 executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB issuing the CREMC macro.
- No linkage is provided between the created ECB and the active segment using this macro.
- There may be up to 104 bytes of data passed to the created ECB's work area.
- The ECB issuing the CREMC macro may be forced into a WAITC if there is insufficient storage available to buffer the parameters. When adequate storage is available, the CREMC macro is executed and control is returned to the next sequential instruction (NSI).
- **The use of this macro should be limited to prevent depletion of storage.**
- Using the PROGRAM parameter sacrifices some performance. Its use should be monitored.
- The program to be activated must have been allocated via the system allocator (refer to *TPF System Installation Support Reference*).
- When activated, the specified program receives control in its allocated addressing mode. The condition code and the contents of registers R0-R7, R14, and R15 are unpredictable.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.

## Examples

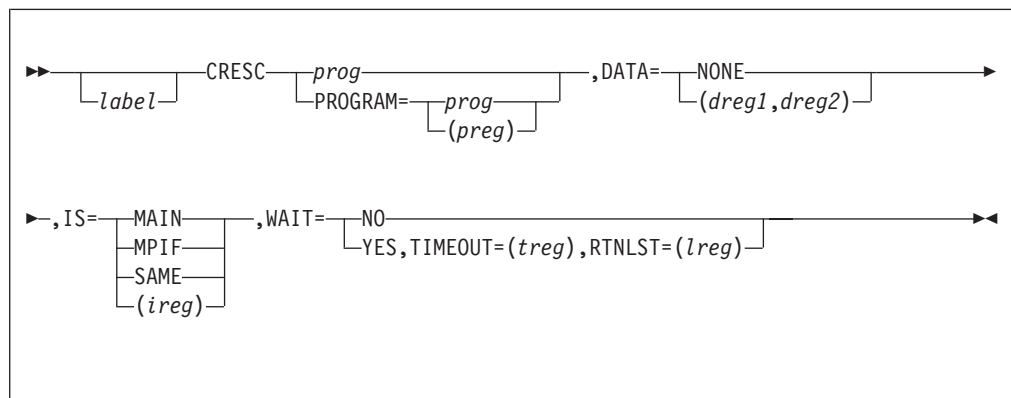
None.

## CRESC—Create New Synchronous ECBs

This general macro creates an entry control block (ECB) for immediate processing by the requested program. The ECB is created in the same subsystem and subsystem user (SSU) as the parent ECB. When exiting, the child ECB will notify the parent ECB by posting an event on which the parent is waiting for completion. Posting the event gives the child ECB the ability to pass the return code value back to the parent ECB.

Use the CRESC macro to have the parent create and then wait for multiple child ECBs to end or time out by coding it multiple times and coding the WAIT=YES parameter on the last entry. This will result in the creation of the child ECBs and will place the parent ECB in a wait state until all child ECBs have been completed or until they time out.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *prog*

The name of the program that will be activated with the created ECB. This is the preferred method for specifying the program name.

#### **PROGRAM**

You can also specify the program name with the PROGRAM parameter. This method has a longer path length than the one described previously.

#### *prog*

The name of the program that will be activated with the created ECB.

#### *preg*

A register (R0–R7) that contains the address of the program name.

#### **DATA**

Specify one of the following:

##### **NONE**

No data will be passed to the new ECB.

#### *dreg1,dreg2*

A number of bytes of data contained in the register specified with *dreg2* will be copied from the address contained in the register specified by *dreg1* and passed to the child ECB and attached to data level 0 (D0). A maximum of 4095 bytes of data can be passed.



**IS** Specifies the I-stream in which the child ECB will run, where:

**MAIN**

The main I-stream.

**MPIF**

The Multi-Processor Interconnect Facility (MPIF) I-stream.

**SAME**

The same I-stream in which the parent is running.

*ireg*

A register (R0–R7) that contains the target I-stream number. At run time, the register must contain the I-stream number on which the ECB is to be created. If the register contains zero, load balancing is used and the ECB is created on the least busy I-stream.

**WAIT**

Specifies one of the following:

**YES**

Create and dispatch the child ECBs and wait for them to be completed.

**NO**

Do not create the child ECBs; just initialize the ECB create information for the child ECBs.

**TIMEOUT=*treg***

Specifies the value that determines how many seconds the parent ECB waits for the child ECBs to be completed, where *treg* is a register that contains a value from 0–32 767.

This is similar to the timeout value on the EVNTC macro. If you specify a value of 0, the parent ECB will wait indefinitely until the child ECBs exit. This value will be in effect in 1052 state.

**RTNLST=*lreg***

Specifies a register that will contain the address of an event block that contains a list of all child ECB return codes on return from the macro, where *lreg* is a register from R0–R7. Field EVNBKL1 will contain the start of the list of all child ECB return codes. The application must set up the child ECB return codes that are used. See the EV0BK DSECT for more information about the event block.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The ECB must be in 31-bit mode.

## Return Conditions

- Control is returned to the instruction following the macro expansion.
- The contents of registers 14 and 15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- Limit the use of this macro to prevent depletion of storage.
- The application can request that a maximum of 50 child ECBs are created with this macro. If more than 50 children are requested, a system error will occur.

## CRESC

- When activated, the specified program receives control in its allocated addressing mode and any data passed will be contained in data level 0 (D0). The condition code and the contents of registers R0–R7, R14, and R15 cannot be predicted.
- In addition to the normal macro trace information, the macro trace for this macro contains the name of the program that will be entered by the new ECB.
- 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

```
* PARENT PROGRAM
* PARENT INITIALIZES THE ECB CREATE INFORMATION FOR 2 CHILD ECBS.

 LA R4,DATA1 FIRST DATA PARM
 LA R5,4 INDICATE LENGTH
 CRESC QXSA,DATA=(R4,R5),IS=MAIN,WAIT=NO

*
* INDICATE THAT PARENT WILL WAIT 100 SECONDS FOR CHILDREN TO
* COMPLETE AND R3 WILL BE USED AS THE RETURN REGISTER.
* R3 WILL POINT TO THE EV0BK BLOCK UPON RETURN.
*
 LA R6,100
 CRESC QXSA,DATA=(R4,R5),IS=MAIN,RTNLST=(R3),WAIT=YES,
 TIMEOUT=(R6)

*
* CHECK RETURN VALUES
*
 EV0BK
 USING EV0BK,R3

 SR R4,R4 INDEX REG
 LA R6,EVBKLI GET ADDRESS OF FIRST ENTRY
 MH R4,EVBKLS GET SIZE OF ENTRY
 AR R6,R4 POINT TO ADDRESS

 USING EVNBKLIF,R6

 CLI EVNBKLIF,EVBK_POST POSTED, WITHOUT ERROR?
 BNE ERROR NO, ERROR
 CLC EVNBKVL5,=F'4' CHECK RETURN VALUE
 BNZ ERROR ERROR IF NOT 4

 DROP R6

 LA R4,1 2ND CHILD
*
* CHECK ERROR BIT IN ENTRY, THEN RETURN VALUE
*
 LA R6,EVBKLI GET ADDRESS OF FIRST ENTRY
 MH R4,EVBKLS GET SIZE OF ENTRY
 AR R6,R4 POINT TO ADDRESS

 USING EVNBKLIF,R6

 CLI EVNBKLIF,EVBK_POST POSTED WITHOUT ERROR?
 BNE ERROR NO, ERROR
```

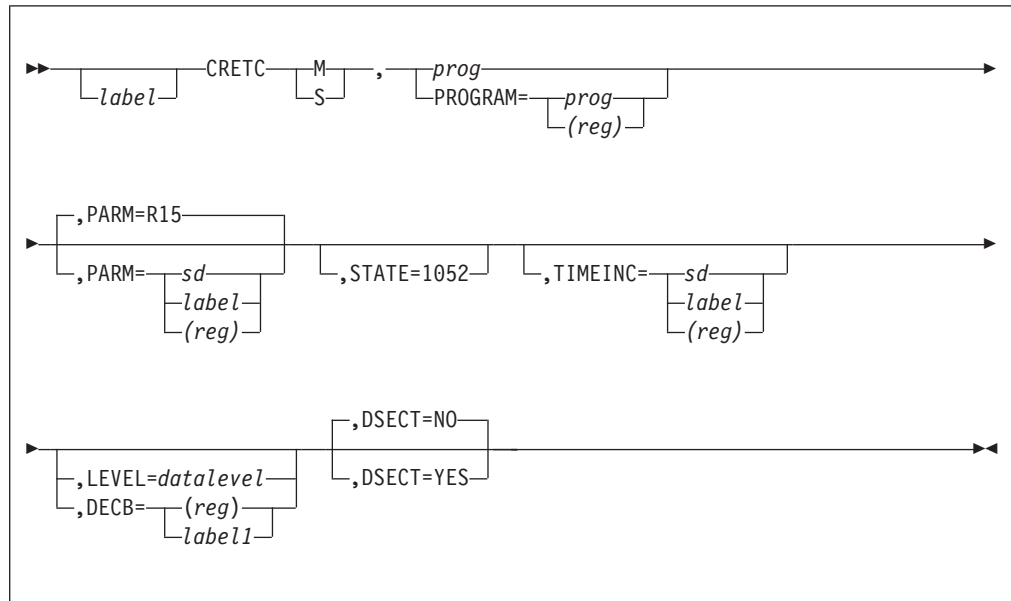
```
CLC EVNBKVL$,=F'4' CHECK RETURN VALUE
BNZ ERROR ERROR IF NOT 4

DROP R6
:
:
* CHILD PROGRAM
* THIS IS HOW THE CHILD ECB EXITS AND RETURNS A VALUE
* TO THE PARENT.
:
LA R1,4
EXITC RC=R1
```

## CRETC—Create a Time-Initiated Entry

This general macro is used to enter a program after a specified interval of time has elapsed. The control program will save this request and transfer control to the specified program at the correct time. A new entry control block (ECB) is created when the specified program is entered. The ECB is created on the same I-stream, in the same subsystem, and subsystem user as the creating ECB.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

**M** The time increment in minutes.

**S** The time increment in seconds.

#### *prog*

The four character alphanumeric name of the program to be activated must be specified either as parameter two or as its keyword alternate. The first character must be alphabetic. This method generates a V-con which is resolved at link edit time into a Program Allocation Table (PAT) displacement. This is the preferred method for specifying the program name.

#### **PROGRAM**

The name of the program can alternately be provided via the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the Program Allocation Table (PAT) displacement. This method has a longer path length than the one described above.

#### *prog*

The name of the program that is to be activated with the created ECB.

#### *(reg)*

A register (R0–R7) that contains the address of the program name.

#### **PARM=sd|label|(reg)**

This parameter specifies the 4-byte action word to be passed to the called

program. The operand may be a self-defining term, a symbolic label, or a register enclosed in parenthesis containing the address of the actual parameter. The default is R15.

**TIMEINC=sd|label|(reg)**

This parameter specifies the number of minutes or seconds from the present time a specified program is to be executed. This parameter is valid only if STATE=1052 has been specified. The operand may be a self-defining term, a symbolic label, or a register enclosed in parenthesis. If not specified the value must be placed in the rightmost 3 bytes (right-justified) of R14.

**Note:** When you specify the time increment in minutes, an ECB is created at a full minute boundary, which can be less than the interval specified with the TIMEINC parameter. For example, assume you specified an interval of 1 minute. If the current time is 10:35:55, the new ECB can be created at 10:36:00.

**STATE=1052**

This optional parameter allows the program specified by *prog* or the PROGRAM parameter to be activated at the time requested even when the system is in 1052 state. The only allowed operand is '1052'. If STATE is not specified, the program requested by *prog* or PROGRAM parameters cannot be activated until the system is cycled above 1052 state.

**LEVEL=datalevel**

This optional parameter allows you to specify a data level (D0...DF) to be passed to the new ECB. The level should be coded as one of the standard data level equates D0, D1, ... DF.

When the new ECB is dispatched, the block from the issuing ECB's data level will be available to the new ECB on its data level 0.

When control is returned to the issuing ECB, the storage block previously attached at the level is detached and is no longer available for use by the issuing ECB.

**DECB=(reg)|label1**

The label or general register (R1–R7) containing the address of the data event control block (DECB) that contains the block to be passed to the created ECB.

When the new ECB is dispatched, the block from the DECB of the issuing ECB will be available to the new ECB on data level 0.

When control is returned to the issuing ECB, the storage block previously attached at the DECB is detached and is no longer available for use by the issuing ECB.

**DSECT=NO|YES**

An optional keyword parameter to request the generation of a DSECT describing the parameter list generated for the CRETC call. The use of this parameter should be limited to the CRETC service routines. If YES is coded, no CRETC expansion will be generated, instead only an expansion of the parameter list is generated. The default is NO.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must contain the time increment that indicates the time relative to the present time when the specified program is to be executed. The time increment must be specified in the rightmost 3 bytes of R14, unless STATE=1052 when the

## CRET

increment can be specified with the TIMEINC parameter. The TIMEINC parameter can only be used when STATE=1052.

- R15 must contain a 4-byte action word that will be passed to the called program. The action word may either be loaded into R15 prior to the execution of this macro, or may be specified with the PARM parameter.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- There is an immediate return to the current program following execution of this macro.
- The contents of R14 and R15 are unknown. The contents of the remaining operational registers and the condition code are saved during execution of this macro.
- If the LEVEL or DECB parameter is specified, the ECB data level or DECB given will be empty on return to the caller and the attached block will no longer be available for use by the current program.

## Programming Considerations

- This macro can be run on any I-stream.
- The ECB reference register (R9) must contain the address of this Entry Control Block being processed before using the macro.
- An ECB is provided when the new program is activated. The address of this ECB is not available to the program executing the CRET macro.
- ECBs created by the CRET macro use the version of the program most recently activated. If this program is incompatible with the program that issued the CRET call, an interface problem may occur.
- The Action Word specified by the PARM parameter (or, by default, the contents of R15) will be stored in the first word of the Entry Control Block (ECB) Work Area One. The address of a working storage block cannot be passed.
- The program name referenced via *prog* or PROGRAM parameters must have been allocated by the system allocator (reference *TPF System Installation Support Reference*).
- If STATE=1052 is specified, the program requested will be activated in all system states, including 1052 state. Otherwise, the request will be honored only when the system has been cycled above 1052 state.
- The ECB issuing the CRET macro with the STATE=1052 option may be forced into an implied WAITC if there is insufficient storage available to buffer the parameters. When adequate storage is available, the CRET macro is executed again and a return is made to the instruction after the macro expansion.
- The use of the STATE=1052 option should be carefully monitored to prevent a depletion of storage.
- Using the PROGRAM parameter sacrifices some performance. Consequently its use should be monitored.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the program being activated.
- 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.

- 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 and the entry is exited.

## **Examples**

None.

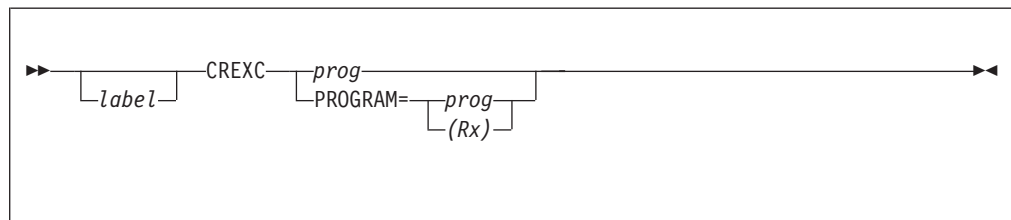
## CREXC—Create a Low Priority Deferred Entry

This general macro creates an independent entry control block (ECB) for deferred processing by the requested program. The ECB is created on the same I-stream, in the same subsystem, and subsystem user as the creating ECB.

A variable number of bytes (0–104) is passed to the created ECB work area. The control program moves the data into an interim block of available storage and adds this block to the deferred processing list. Operational Program Zero will initialize an ECB with the data in the work area, release the interim block, and execute an Enter with No Return (ENTNC) to the specified program.

CREXC is a specialized macro very similar to the CREDC Macro. The difference is that CREXC requires that a larger number of storage blocks be available to allow execution. It is recommended that the CREXC macro be used by programs that create many entries. This will ensure normal system operation without impairment, which could be caused by a depletion of storage blocks.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be activated with the created ECB. This method generates a V-CON which is resolved at link edit time into a Program Allocation Table (PAT) displacement. This is the preferred method for specifying the program name.

#### PROGRAM

The name of the program can alternately be provided using the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the PAT displacement. This method has a slightly longer path length than the positional parameter method.

*prog*

The name of the program that is to be activated with the created ECB.

*(Rx)*

A register (R0-R7) that contains the address of the program name.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must contain the number of bytes of parameters to be passed to the created ECB work area.
- R15 must contain the address of the start of the parameters which are to be passed.



## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The remaining operational program registers are saved during the execution of the CREXC macro.

## Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB issuing the CREXC macro.
- No linkage is provided between the created ECB and the active program segment using this macro.
- There may be up to 104 bytes of data passed to the created ECB's work area.
- The CREXC macro should be used only by utility-type programs running in the system. The reason for this is that when a large number of system entries are being created, the system efficiency can be lowered through depletion of storage blocks. The CREXC macro functions like a CREDC except that it ensures a greater number of blocks available for real-time operations within the system.
- The ECB issuing the CREXC macro may be forced into a WAITC if there is insufficient storage available to buffer the parameters. When adequate storage is available, the CREXC macro is executed and control is returned to the next sequential instruction (NSI).
- The use of this macro should be monitored in order to prevent a depletion of storage.
- The program to be activated must have been allocated via the system allocator (refer to *TPF System Installation Support Reference*).
- When activated, the specified program receives control in it's allocated addressing mode. The condition code and the contents of registers R0-R7, R14, and R15 are unpredictable.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.

## Examples

```
CREXC PROGRAM=MINE
```

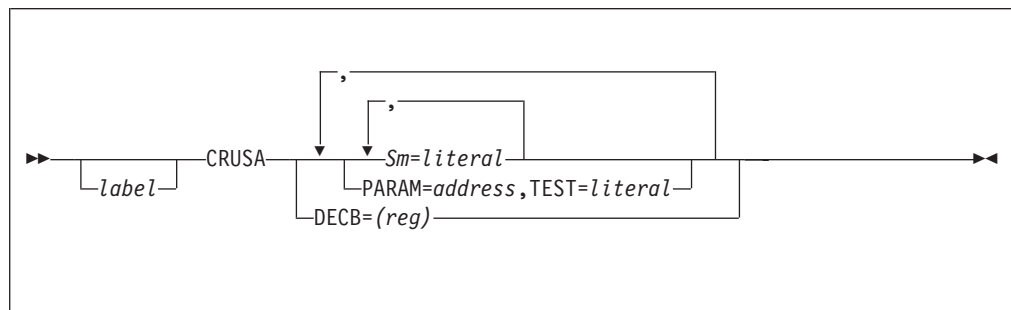
This call to CREXC activates the MINE program.

## CRUSA–Test and Release Data Level

Use this general macro to do any of the following:

- Test and release ECB data levels for core blocks being held
- Test an ECB data level and branch to a specified address if the ECB data level is not holding a core block
- A combination of the first two functions on the list
- Test a data event control block (DECB) for a core block being held and release the core block if it is present.

## Format



*label*

A symbolic name can be assigned to the macro statement.

**TEST=literal**

A literal specifying a data level to be tested for a core block. The literal is a hexadecimal digit  $0 \leq (\text{literal}) \leq F$ . The default is NONE. The TEST parameter is used in conjunction with the PARAM parameter.

**PARAM=address**

If no core block is held on a level specified by the TEST parameter, control is transferred to the address specified by PARAM. If the data level (specified by the TEST parameter) holds a core block, control continues with the next sequential instruction.

**Sm=literal**

The literal specifies the data levels to test for core blocks being held. Any core blocks found are released. At least one level (specified as a hexadecimal digit from 0 to F) must be specified; if more than one level is specified, they can be in any order. When specifying *m*, start with 0, 1, 2, ... 15 in order. Note that *m* is an incremental parameter, not a data level.

For example, data level 6 alone can be tested and released with

`S0=6`

and data levels 3, 5, and 7 can be tested and released with

`S0=3,S1=5,S2=7`

**DECB=(reg)**

The general register (R1–R7) containing the address of the DECB to test for a core block being held. If a core block is found, it is released. Only a single DECB can be processed at one time.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- If the TEST parameter is used, the PARAM parameter must also be present, and vice versa.
- The caller must specify a list of ECB data levels or a single DECB to be tested, but not both.

## Return Conditions

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- If the TEST option is not used, control is returned to the next sequential instruction.
- All released core storage reference words are initialized to indicate that a block of storage is no longer held.
- If the TEST option has been used alone, the data level specified as a literal is not released if one is held. Whether or not a core block is held, control is returned to the next sequential instruction.
- The condition code is changed by the LEVTA macro.

## Programming Considerations

- This macro can be run on any I-stream.
- The appropriate core block reference words (CBRWs) are initialized to indicate a block of storage is no longer held at those ECB data levels or a DECB.
- The LEVTA macro should be used when a positive or negative return, or both, are required.
- CRUSA uses the macros LEVTA and RELCC.
- The Sm, TEST, and PARAM parameters may be used together. Care should be taken that the same data levels are not specified by TEST and Sm. The data levels specified by Sm are tested prior to the data level specified by TEST.
- 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

- Test and release data level 1  
CRUSA S0=1
- Test and release data levels 6 and 3  
CRUSA S0=6,S1=3
- Test and release data levels 1, 2, 3, 4, 5, 6, 7, 8, and 9  
CRUSA S0=1,S1=2,S2=3,S3=4,S4=5,S5=6,S6=7,S7=8,S8=9
- Test data level 1 and branch to label CSLD055 if no core block is held  
CRUSA TEST=1,PARAM=CSLD055
- Mixing Sm, Test, and PARAM specifications  
CRUSA S0=3,S1=5,TEST=4,PARAM=NOHOLD4

specifies that data levels 3 and 5 are tested and released if they hold core blocks, and if a data level 4 test indicates no core block is held, control transfers to the label NOHOLD4, otherwise control resumes with the next sequential instruction.

## CRUSA

- Test the DECB whose address is stored in R7 and release the core block if it is present.

CRUSA DECB=(R7)

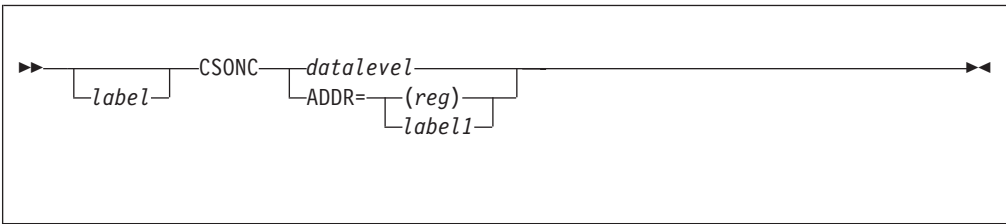
# CSONC—Convert System Ordinal Number

This general macro converts a file address reference format (FARF) format address to a 7-byte file address in the MMCCHHR format. The name MMCCHHR displays how the address is constructed:

- MM     Halfword symbolic module number used to identify the physical device
- CC     Halfword cylinder address
- HH     Halfword head address
- R       1-byte record number.

The 5-byte CCHHR is the same as the record ID recorded in the count area on file.

## Format



*label*  
A symbolic name can be assigned to the macro statement.

*datalevel*  
A file address reference word (FARW) in the range D0–DF.

**ADDR=(reg)|*label1***  
This parameter specifies the general register (R1–R7) containing the location, or a label indicating the location, of an 8-byte file address.

## Entry Requirements

- If an ECB data level is coded, R9 must contain the address of the entry control block (ECB) being processed.
- If an ECB data level is coded, the specified FARW must contain a FARF address.

## Return Conditions

- Control is returned to the next sequential instruction.
- R14 contains the 2-byte module (MM) and cylinder (CC) numbers.  
R15 contains the 2-byte head number (HH) and the 1-byte record number (R).  
The rightmost byte of R15 contains one of the following hexadecimal device type codes:  

| Code  | Symbolic Device Type |
|-------|----------------------|
| X'0C' | DEVA                 |
| X'10' | DEVB                 |
| X'14' | DEVC                 |
| X'18' | DEVD                 |
- The contents of R0–R7 are saved during execution of this macro.

## CSONC

- Return codes in R14 indicate compatibility of FARF addresses.
  - R14 is negative when the given FARF address is compatible with the migration stage, but is not in the proper bounds. For instance a FARF4 address input in the FARF3/4 stage does not have a defined UFT/FTI combination.
  - R14 contains a return code if the FARF6 address does not decode.
  - When R14 is not negative, this implies satisfactory execution and the return of valid values in R14 and R15.
- 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 or the location referenced by the ADDR parameter is unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- R14 should be interrogated for a negative result, denoting an invalid FARF address given as input to this macro.

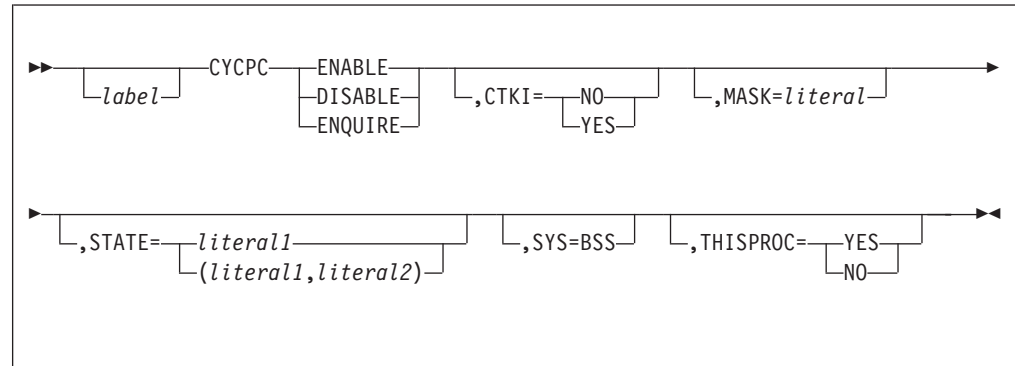
## Examples

None.

## CYCPC–PSMS Utility Interface Macro

This general macro provides the interface between the Processor Status Management Services package (PSMS) and programs requiring the ability to inhibit or allow the system to cycle and those requesting the status of the system state.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **ENABLE**

System cycle is to be allowed.

#### **DISABLE**

System cycle is to be inhibited.

#### **ENQUIRE**

System status is requested.

#### **CTKI**

Keypoint CTKI retrieval option (optional)

#### **NO**

Indicates keypoint CTKI will be retrieved, if required, on level 8 (Default)

#### **YES**

Indicates that keypoint CTKI will be passed to the PSMS package on level 8

#### **MASK=literal**

Required if ENABLE or DISABLE is coded.

This parameter is a preassigned value which identifies the calling utility. It is ignored if ENABLE is coded.

#### **STATE**

Required if DISABLE or ENQUIRE is coded. It is ignored if the ENABLE option is coded.

This parameter identifies the system state addressed by the DISABLE or ENQUIRE option.

*literal1*

Identifies the system state being tested. Specify one of the following:

- 1052
- UTIL
- CRAS
- MESW

## CYCPC

- NORM.

### *literal2*

Identifies the requested status of all processors. Specify one of the following:

- ABOVE
- BELOW.

### **SYS=BSS**

This is optional.

If coded, this parameter must equal "BSS". It is required when it is called by a program that resides in the BSS. It identifies the path that will be taken in activating the PSMS routine.

### **THISPROC**

This specifies if this processor is to be included in the test for the status of all processors.

#### **YES**

Include the system state of this processor in determining if the requested condition is satisfied (Default).

#### **NO**

Exclude the system state of this processor in determining if the requested condition is satisfied.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- ECB level 8 must be available for use by the PSMS package if CTKI=NO is coded (or the default is allowed).
- ECB level 8 must address CTKI for use by the PSMS package if CTKI=YES is coded.
- ECB workarea EBW001-EBW005 is used by the PSMS package.
- ECB field EBSW01 is used by the PSMS package.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of ECB workarea EBW001-EBW005 will have been altered.
- On return, ECB field EBSW01 will be set as follows:
  - Inhibit request (DISABLE)
    - X'00'** Cycle successfully inhibited.
    - X'FF'** Cycle unsuccessfully inhibited.
    - X'01'** Cycle already inhibited.
  - Allow request (ENABLE)
    - Not altered.
  - Status request (ENQUIRE)
    - X'00'** All processors are in the specified state.
    - X'FF'** All processors are not in the specified state.

## Programming Considerations

- This macro can be executed on any I-stream.



- Masks identifying utility packages and used for the MASK parameter must have been preassigned in the PSMS package.
- The cycle 'DISABLE' indicators as set by this function are enforced until cancelled by a subsequent cycle 'ENABLE' request.
- The SYS keyword must not be coded if the activating program is not BSS resident.
- An incorrectly coded value for 'literal1' in the STATE keyword will result in an UNDEFINED SYMBOL assembly error.
- An incorrectly coded value for the 'MASK' keyword will result in an UNDEFINED SYMBOL assembly error.

## Examples

Disable cycle above 1052 state on all processors to allow the Tape Loader utility to process.

```

Label1 CYCPC DISABLE, X
 STATE=(1052,BELOW), X
 MASK=EPSTPLDR
 TM EBSW01,X'FE' TEST EBSW01 FOR RETURN CODE
 BZ ALLOK ALL PROCESSORS AT 1052
 B NOTOK ALL PROCESSORS NOT AT 1052
ALLOK DS 0H
*
*
* TAPE LOADER PROCESS
*
* ENABLE THE CYCLE FUNCTION
 CYCPC ENABLE, X
 MASK=EPSATIM X

```

Check the status of other processors before starting a utility on this processor.

```

Label1 CYCPC ENQUIRE, X
 STATE=NORM, X
 THISPROC=NO
 TM EBSW01,X'FF' TEST EBSW01 FOR RETURN CODE
 BZ ALLOK ALL OTHER PROCESSORS IN NORM
 B NOTOK ALL OTHER PROCESSORS NOT IN NORM
ALLOK DS 0H
*
*
* RUN UTILITY

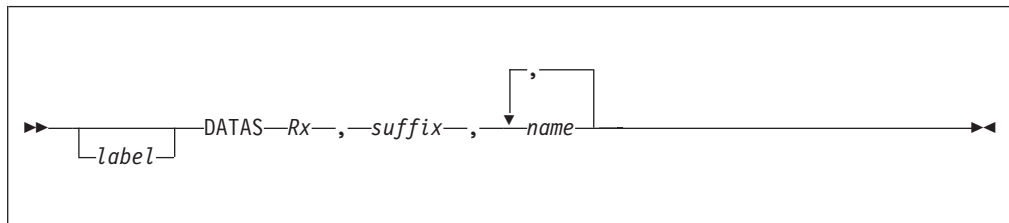
```

### DATAS–Combination Data

This general macro allows as many as 8 data records to be issued using the same base register by creating 1 combination data macro.

At times, 2 or more data macros refer to a single data record and it is often necessary that only one base register be used. One example of this is an input message (AM0SG) located in an output message (UI0OM). Because of the philosophy of the assemblers, 2 separate data macros cannot be issued with USING statements for the same registers. The DATAS macro can be issued to resolve this problem.

### Format



#### *label*

A symbolic name can be assigned to the macro statement. It must be used if you wish to reissue a USING statement.

*Rx* This is a required parameter. The valid registers are R1-R7, R14, R15.

#### *suffix*

This is a required parameter. It defines a one-character suffix to be used.

#### *name*

This is the name of a data macro. Up to eight data macros may be specified.

### Entry Requirements

- The records expanded in this macro must be suffixed.
- To reissue a USING statement, the DATAS macro must be labeled and the USING statement will reference this label.
- A suffix character cannot be used more than once in any assembly for any particular data macro.
- Any data macro included in a combination macro can be defined with another register by using no suffix or a different suffix.
- When more than one record type is processed in a program and these records share the same register, it is advisable that single data macros be issued just prior to the code that deals with this record type. The combination macro should be used only when the same record is defined by more than one data macro.

### Return Conditions

All the specified data macros will have been issued with the suffix and the same base register.

### Programming Considerations

- This macro is limited to those data macros identified in DATAS source code.
- The specified register will be the base register for all the data macros issued.

- To reissue a USING statement, DATAS must be labelled, and the USING statement will reference this label.
- The same suffix character cannot be used more than once in any assembly for any one particular data macro.
- DATAS can be used more than once within the same program. The restrictions on DATAS are the same as those on noncombination data macros:
  - The same suffix cannot be used more than once in any assembly for any one particular data macro.
  - Two separate data macros cannot be issued with USING statements for the same register.
  - The DATAS macro may be used on any I-stream.

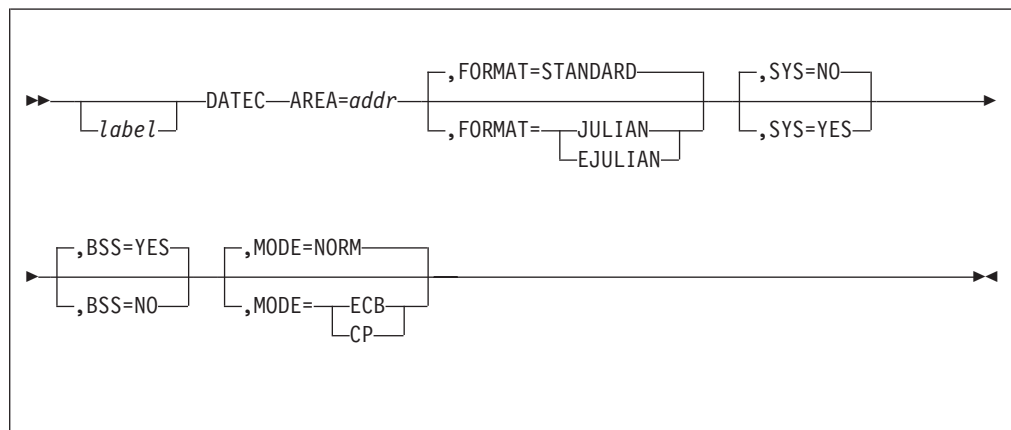
## Examples

```
DATAS R1,X,UI00M,AM0SG
```

## DATEC—Compute Date Stamp

This general macro stores a date in the storage area specified by the caller. The date will be returned in one of several forms, as specified by the caller.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **AREA=addr**

The address of the storage area in which the date is to be placed. The address can be any valid assembler expression. If used, the base or index registers must be in the range 0-9.

#### **FORMAT**

The pattern of the date that is to be returned. Acceptable values are:

##### **STANDARD**

An 8-byte date is to be returned in the format YYYYMMDD.

##### **JULIAN**

A 5-byte date is to be returned in the pattern YYDDD, where YY is the last two digits of the year and DDD is the number of days so far this year, including the current day (e.g., 27 August 1989 would be 89239).

##### **EJULIAN**

A 6-byte date is to be returned in the pattern CYYDDD, where C is an indicator of the thousands and hundreds positions of the year (X'40' = 19, "0" = 20, "1" = 21, etc.), YY is the last two digits of the year, and DDD is the number of days so far this year, including the current day.

#### **SYS**

A system or subsystem date is returned. Acceptable values are:

##### **YES**

The system local standard date is to be returned. This is the default.

##### **NO**

The subsystem local date is to be returned.

#### **BSS**

The subsystem for which the subsystem local date is to be returned. This parameter is valid only when a CP expansion is generated and SYS=NO is specified or defaulted. Acceptable values are:

**YES**

The subsystem local date is to be returned for the BSS.

**NO**

The subsystem local date is to be returned for the subsystem whose index is contained in R15.

**MODE**

The kind of macro expansion to generate. Acceptable values are:

**NORM**

The type of expansion is to be determined based on the value of a global variable that is set when the BEGIN macro is invoked.

**CP**

A CP expansion is to be generated.

**ECB**

An ECB expansion is to be generated.

## Entry Requirements

- If BSS=NO is specified then R15 must contain the subsystem index of the subsystem whose local date is to be returned.
- For ECB expansions, R8 must point to the program base and R9 must point to the ECB.

## 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 protect key is unchanged.
- The date is returned in the specified area, in the specified format.

## Programming Considerations

- The EJULIAN date format conforms to the format used for IBM standard tape labels.
- This macro can be executed from the CP or from an ECB program. When executed from an ECB program, fast-link macro linkage is used.
- This macro can be executed on any I-stream.
- The source of the date returned by this macro is only updated when the system is above 1052 state.
- The date returned by this macro is only valid if clock restart has completed; no checks are made by this macro for completion of clock restart.

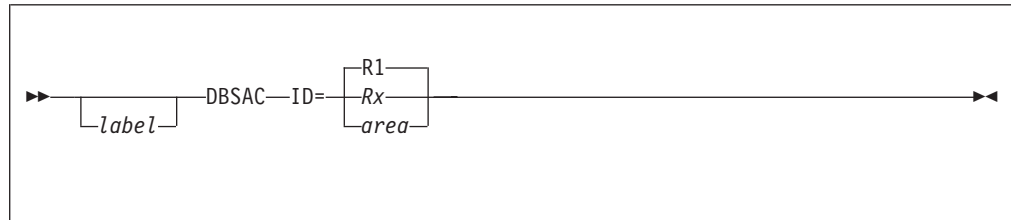
## Examples

None.

## DBSAC—Attach TPF Application Requester Database Support Structure

This general macro attaches the TPF Application Requester (TPFAR) feature Structured Query Language Transaction Profile (STP) and associated blocks to an entry control block (ECB). This database support structure must have been detached from this or another ECB using the DBSDC macro. See “DBSDC—Detach TPF Application Requester Database Support Structure” on page 136 for additional information.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

**ID** Specify one of the following:

#### *area*

The symbolic name of a fullword area that contains the address of an 8-byte identifier for the TPFAR database support structure to attach. The TPFAR database support structure identifier must have been returned by DBSDC.

#### *reg*

A register, R0–R5, containing the address of a fullword area that contains the address of the identifier for the 8-byte TPFAR database support structure to attach. The TPFAR database support structure identifier must have been returned by DBSDC.

If the ID parameter is omitted or invalid, R1 is the default.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A TPFAR database support structure must have been detached using the DBSDC macro.
- A TPFAR database support structure is not already attached to the ECB.

### Return Conditions

- Control is returned to the next sequential instruction.
- R6 contains one of the following return conditions:
 

|                      |                                                                            |
|----------------------|----------------------------------------------------------------------------|
| 0 - DBSAC_SUCCESSFUL | Completed successfully.                                                    |
| 1 - DBSAC_INUSE      | A TPFAR database support structure is already attached to the current ECB. |
| 3 - DBSAC_DBSFINDERR | Unable to find the TPFAR database support structure.                       |

4 - DBSAC\_CCAFINDERR      Unable to find a file record associated with a cursor control area (CCA) entry.

- The contents of R7, R14, and R15 are unknown. The contents of all other registers are preserved across this macro call.

## **Programming Considerations**

- This macro can be executed on any I-stream.
- A TPFAR database structure must have been released using the DBSDC macro.
- A system error with return results if an invalid TPFAR database structure identifier is specified.
- When the TPFAR database structure is reattached to the ECB, the TPFAR database structure identifier is no longer valid and must not be reused.

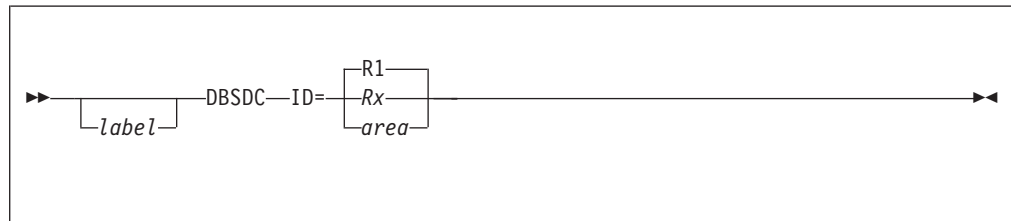
## **Examples**

None.

## DBSDC—Detach TPF Application Requester Database Support Structure

This general macro detaches the TPF Application Requester (TPFAR) feature Structured Query Language Transaction Profile (STP) and associated blocks from an entry control block (ECB). The ECB must have issued structured query language (SQL) requests. See “DBSAC—Attach TPF Application Requester Database Support Structure” on page 134 for additional information.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

**ID** Specify one of the following:

#### *area*

The symbolic name of a fullword area containing the address of an 8-byte contiguous area where the TPFAR database support structure identifier is returned.

#### *reg*

A register, R0–R5, containing the address of a fullword area containing the address of an 8-byte contiguous area where the TPFAR database support structure identifier is returned.

**Note:** The TPFAR database support structure identifier is needed to reattach the TPFAR database support structure.

If the ID parameter is omitted or invalid, R1 is the default.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A TPFAR database support structure must be attached to the ECB.
- SQL requests must have been issued from the ECB.

### Return Conditions

- Control is returned to the next sequential instruction.
- R6 contains these return conditions:
 

|                      |                                                             |
|----------------------|-------------------------------------------------------------|
| 0 - DBSDC_SUCCESSFUL | Completed successfully.                                     |
| 1 - DBSDC_NONE       | No TPFAR database support structure is attached to the ECB. |
- The contents of R7, R14, and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The specified area contains the TPFAR database support structure identifier needed to reattach the TPFAR database support structure later.



## Programming Considerations

- This macro can be executed on any I-stream.
- SQL requests must have been previously issued from the ECB.
- The TPFAR database support structure identifier returned is needed to reattach the TPFAR database support structure later.
- If the TPFAR database support structures are not reattached using the DBSAC macro, conversations with the remote database are lost until the next IPL.
- The DBSDC macro uses short term pool files to save the database support structure. The DBSAC macro must be issued before recycling the short term pool directories.
- A system error with return results if no TPFAR database support structure is associated with the current ECB.

## Examples

None.

---

## **DDATA—Terminal Data Field Display**

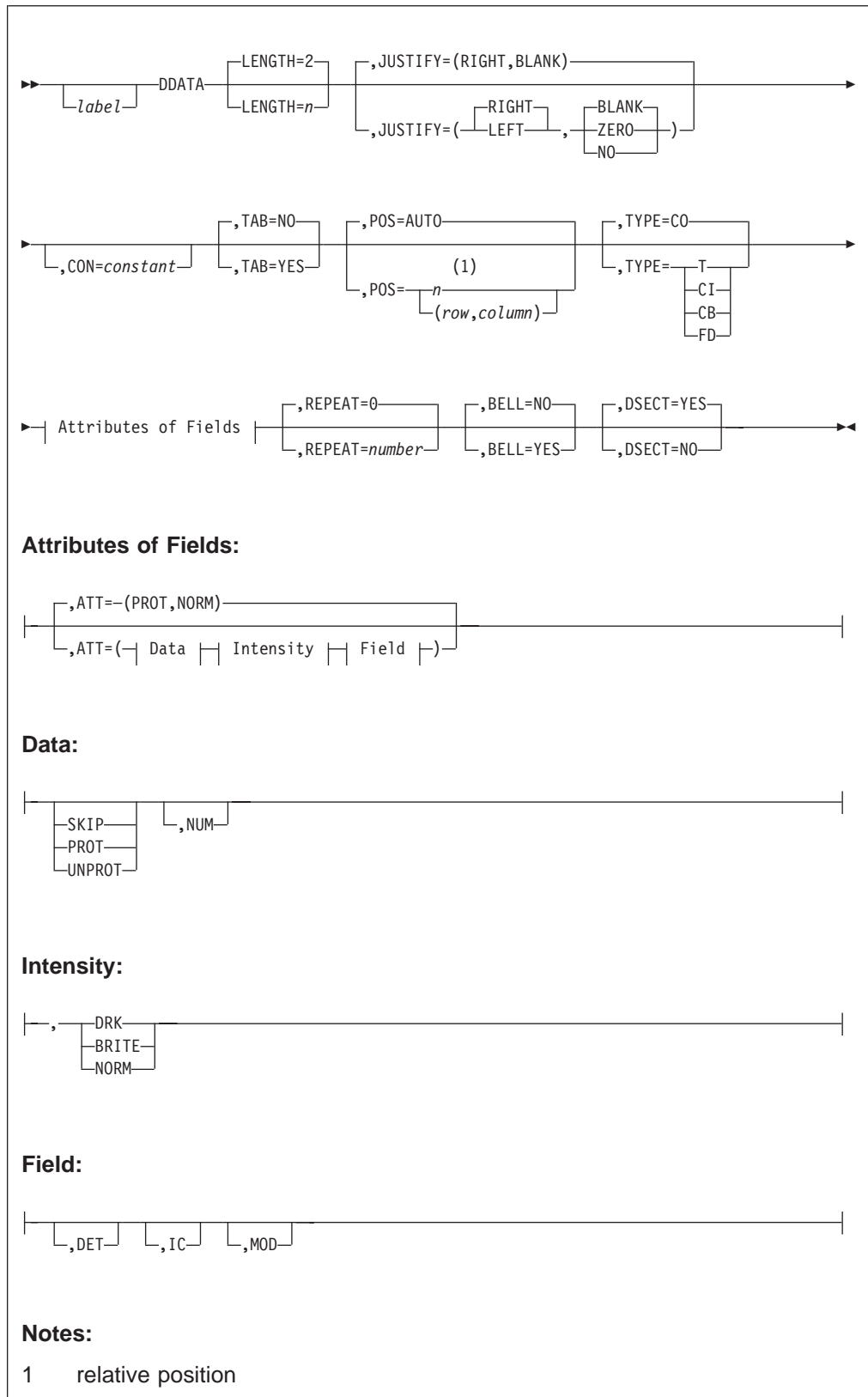
This general macro instruction, with the DPANL macro instruction, is used in application development to declare a data structure used in processing terminal displays. DPANL and DDATA macro instructions (called map statements) that are written by an application programmer must be placed on a map source statement library. Map statements are then included in ECB-controlled source code by using the assembler language COPY instruction. The copied map statements have the effect of declaring symbolic names used in a display data structure, because the DPANL and DDATA instructions expand into a DSECT included in the source code. The DSECT is adjusted and a register is assigned as the result of parameters included in the map statement coding.

The use of map statements is analogous to the use of a data macro instruction. However, map statements do not appear in the source code of ECB-controlled programs. This means that source code does not contain the register assignment of the DSECT generated as the result of copied map statements.

A map statement on the map source library is used in two different ways:

- A map statement is copied into ECB source code as outlined above. Specifically, the DDATA macro instruction is used to define the length of each data field comprising a display.
- A map statement is used by a portion of the mapping support package to create a physical terminal map record (AS0MP) in the fixed file area. These records are used by the system supplied ECB-controlled programs to superimpose or eliminate control characters in data streams processed by application programs. Specifically, the DDATA macro instruction is used to define the position and characteristics (attributes) of each data field comprising a display.

# Format



*label*

## DDATA

A one to four character symbolic name (alphanumeric characters) assigned to this field definition. This symbolic name, if provided, must not appear on any other DDATA statements in this specific application program assembly. If no name is written for this field item, the user will not be able to symbolically reference this field within the assembled program.

Each DDATA item provided with a symbolic reference generates the following DSECT items when the user program is assembled:

- When an input format has been indicated (MODE=IN or MODE=BOTH):

|         |    |          |                         |
|---------|----|----------|-------------------------|
| SYMBOLL | DS | H        | ACTUAL LENGTH OF DATA   |
| SYMBOLI | DS | CLnumber | ACTUAL INPUT DATA FIELD |

- When an output format has been indicated (MODE=OUT or MODE=BOTH):

|         |    |          |                         |
|---------|----|----------|-------------------------|
|         | DS | 0H       |                         |
| SYMBOLA | DS | CL1      | ATTRIBUTE OVERRIDE BYTE |
|         | DS | CL1      | RESERVED                |
| SYMBOLO | DS | CLnumber | OUTPUT DATA FIELD       |

**Note:** The appended suffixes are 'L' and 'I' for input fields and 'A' and 'O' for output fields.

The original Symbol and the appended suffix yield the generated symbolic field name.

### DDATA

These five (5) characters represent the macro name used to define individual fields comprising a format. This must appear in the field statement.

#### LENGTH=*n*

This optional keyword denotes the data or text length of an input or output field. The maximum value is LENGTH=255. The default length is 2. When defining the length of either an output or input field, this parameter will exclude the length of the attribute character. Thus, when LENGTH=6 is coded, the following DSECT items are generated for a field defined as output:

|         |    |          |  |
|---------|----|----------|--|
|         | DS | 0H       |  |
| SYMBOLA | DS | CL1      |  |
|         | DS | CL1      |  |
| SYMBOLO | DS | CLnumber |  |

For fields defined as input, LENGTH=6 generates the following DSECT items:

|         |    |     |  |
|---------|----|-----|--|
| SYMBOLL | DS | H   |  |
| SYMBOLI | DS | CL6 |  |

#### JUSTIFY=(LEFT|RIGHT,BLANK|ZERO|NO)

This optional keyword enables the user to specify, for both input and output data fields, whether the actual data is to be left or right justified. In addition, the user may indicate whether the field (input or output) is to be preset with blanks or numeric zeroes. The default value for the first subparameter is RIGHT, and, the default value for the second subparameter is BLANK. When this keyword is omitted, JUSTIFY=(RIGHT,BLANK) is assumed.

#### CON=*constant*

This optional keyword provides the capability for the user to define, on output only, any constant data to be displayed on a 3270 CRT terminal or hardcopy device. This Keyword is not valid for 1977 type devices. The maximum number of character positions allowed, including blanks, is 255. When the constant information contains imbedded blanks, the entire field must be enclosed with quotes.

CON='SIGN IN'

When no blanks are included in the data stream, the quotes can be omitted.

CON=PAYROLL

**TAB=YES|NO**

This optional keyword parameter is only valid for output maps. Printer devices with the mechanical tab feature (1980 Model 24 and 1977 with tab) may have fields positioned at specific locations on output by specifying TAB=YES. The utilization of this particular feature requires that the user manually set the column position on the device prior to referencing this map. If this keyword is omitted, TAB=NO is assumed.

**POS**

This keyword provides the location of a specific input or output field within a format. For both input and output, this is the location where the attribute character appears; the actual text follows the attribute character. For non-3270 type devices, POS represents the actual location of a field. The user may specify one of three parameters with this keyword; relative position, row-column, or automatic position generation. Once one is selected, it cannot be changed on subsequent DDATA statements.

*relative position*

The user denotes a location, relative to position 0, of the data. Thus, POS=10 indicates data is to be input or output at the tenth position, relative to zero.

*(row,column)*

The user denotes a location by specifying a row number and a column number where data is to be input from or output to. Thus, POS=(1,1) represents the position on a 3270 CRT or hardcopy printer of a data field. Both row and column are relative to one, not zero.

**AUTO**

The Mapping Support Package generates the location and addresses of the field definitions comprising a specific map format. The positioning of the various fields is dependent upon the 'TYPE' parameter. The field classifications (TYPE=) affect their respective locations on both input and output.

The absence of the POS= operand from the DDATA statements will cause a default to the automatic (AUTO) field generation feature.

**TYPE**

This optional keyword, which is used in conjunction with the AUTO feature of map generation, defines or classifies the nature of a specific field, whether for input or output formats. The following field types have been designated as valid for this keyword. If this keyword is omitted, TYPE=CO is assumed.

**T** This designation indicates the defined field is a title or header field for a specific panel. This field will be centered on a single row; the user must insure that the length of this field does not exceed the number of character positions per line for the specified device. This value is only valid for output maps.

**FD**

The field descriptor definition (FD) positions this field one position past the end of the previous field. A field defined as TYPE=FD will not be split over two lines; if the entire field cannot be contained in the remaining portion of this line, it is positioned at the beginning of the next line. Fields designated as FD should not be included in the MAXFLDS count of the DPANL statement. This value is only valid for output maps.

**CO**

This designation specifies a field as common output (CO). It is to be used to denote a field that is to appear in an output map only. The field so designated (TYPE=CO) will be positioned two spaces beyond the previous field and extend over as many lines as needed to contain the field.

**CI**

This designation specifies a field as common input (CI). It denotes a field that is to appear in an input map only. The field positioning is the same as TYPE=CO.

**CB**

This designates a single field as appearing on both input and output maps. Field positioning is identical to TYPE=CI and TYPE=CO fields.

**ATT=SKIP|PROT|UNPROT,NUM,BRITE|DRK|NORM,DET,IC,MOD**

This optional Keyword is used to specify the device dependent characteristics and attributes applicable to individual fields. This keyword is valid only for output maps and for 3270 devices. When no parameters are specified, a field is assumed to have the following characteristics:

protected,  
alphameric,  
displayable at normal intensity,  
nonselector pen-detectable and not modified.

- SKIP, PROT, UNPROT, and NUM Attributes

These attributes are used to describe the capability of a field to receive data. The first, second, and third items are mutually exclusive; only one may be selected. When PROT is specified, data cannot be keyed into the field. When SKIP is specified, the cursor automatically skips over the field and sets itself at the start of the next unprotected field. A field designated as SKIP cannot have data keyed into it. The UNPROT attribute allows data to be keyed into a field.

If the DPANL statement indicates SCREEN=SPLIT, and UNPROT is specified as an attribute, the map will be terminated, as these two conditions are incompatible.

When ATT=NUM is specified, the data entry keyboard is set to numeric shift for this field.

- BRITE, NORM and DRK Attributes

These attributes are mutually exclusive, designate a field as high intensity, normal intensity, and nondisplay/nonprint, respectively.

- DET Attribute

Specifies that a field may be pen-detectable. For a field designated as pen-detectable, the first data character must be a '?', a '>', or a blank. Reference the IBM 3270 Information Display System Component Description for a more detailed explanation of pen-detectable fields. An input DET field has an associated one byte reserved data area which is set to X'00' when the field is unselected or X'FF' when the field is selected.

If PROT is specified with DET, bit 5 of AS1MNT (see AS0MP) will be set on to indicate unprotected fields are specified. This will avoid inhibition of data on the screen.

- Insert Cursor (IC) Attribute

Indicates that the cursor is to be placed at the first position of this field. When ATT=IC is specified for more than one field in a given map, the address of the last specified field is used for cursor placement. The absence of the IC parameter denotes no change to the cursor position; it will remain at the same screen position it previously was situated at.

- **MOD Attribute**

Indicates that this field should have the modified data tag (MDT) set on when the field is sent to the 3270. On a subsequent read from this terminal, the field is treated as if it had been modified and will be read in. This facility provides the means to read constant information (see CON=) from the screen as input.

The following attributes are valid for output maps only:

- SKIP
- DRK
- PROT
- NORM
- UNPROT
- IC
- NUM
- MOD
- BRITE

DET is valid only for input maps.

**REPEAT=number**

This optional keyword indicates that the field defined with this DDATA statement is to be repeated on input or output up to a maximum of 255 times. Each repeated field will be positioned in the same column as the previous one, but on the next line (row). When the REPEAT parameter has been specified, no other field may be positioned on the same line (row) as that of the repeated field. In the EXAMPLE section there is an example depicting the field suffixing related to a REPEAT field. This option is not applicable to the 3287 SLU printers. If this keyword is omitted, then REPEAT=0 is assumed and the field will not be repeated.

**BELL=YES|NO**

This parameter causes an alarm or similar attention device to be sounded at the printer. The BELL code is placed prior to the field text in which it was defined in the output data stream.

The default is BELL=NO.

This parameter is only valid for maps whose definition causes an LU type 1 (SCS) data stream to be formatted. The devices that support this feature are the 3287 Models 1, 2, and 4 SLU devices.

**DSECT**

This optional keyword is valid only for output maps. This keyword can be used to exclude a field from the map being created.

**NO**

The field (described by this DDATA macro) will be excluded from the map.

**YES**

The field will be included in the map.

If this keyword is omitted, DSECT=YES is assumed.

## **DDATA**

### **Entry Requirements**

DDATA statements and their associated DPANL statements are placed by the user on a symbolic map library under the unique 1–4 character map name used on the DPANL statement. The user references a specific map definition in his program by issuing a COPY 'mapname' statement in the source code.

### **Return Conditions**

None.

### **Programming Considerations**

The DDATA macro may be used on any I-stream.

### **Examples**

None.

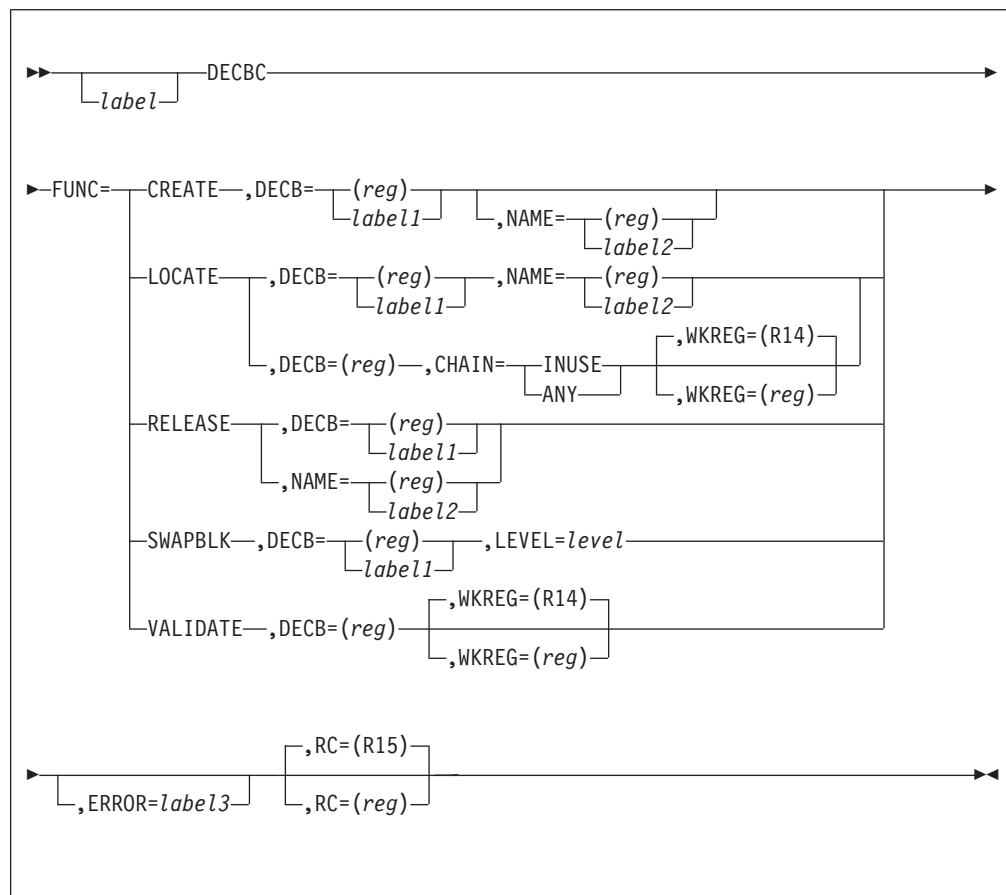


## DECBC—Manage Data Event Control Blocks

Use this general macro to perform the following actions:

- Create a new data event control block (DECB)
- Locate an existing DECB
- Swap a core block between a DECB and an entry control block (ECB) data level
- Release an existing DECB that is no longer in use
- Determine if a given address references a valid DECB that is in use.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### FUNC

Specifies the requested DECB action. The following values are allowed:

##### CREATE

Create a new DECB.

##### LOCATE

Locates a previously allocated DECB with a name matching the value specified in the NAME parameter or locates the next DECB in the chain specified by the CHAIN parameter.

##### RELEASE

Releases the DECB referenced in the DECB or NAME parameters (if the DECB is not currently in use).

## DECBC

### SWAPBLK

Exchanges core block and related information between a DECB and a specified ECB data level.

### VALIDATE

Determines if the DECB specified is a valid DECB.

### DECB=(*reg*)/*label1*

For FUNC=CREATE and FUNC=LOCATE, specifies the label or general register where the address of the DECB is to be returned. For all other actions, specifies the label or general register containing the address of a valid DECB for which the specified action is to be performed. For E-type programs, allowable register values are R1–R7. For control programs (CPs), allowable register values are R1–R7, R14, or R15.

### NAME=(*reg*)/*label2*

The general register containing the address (or a label indicating the address) of a 16-byte name associated with a DECB. For E-type programs, allowable register values are R0–R7. For CPs, allowable register values are R0–R7, R14, or R15.

### CHAIN

Specifies the next DECB entry to be returned.

### INUSE

Returns the next active DECB chained to the current DECB. The register specified by the DECB parameter indicates the address of the current DECB. If the register specified by the DECB parameter contains a value of zero, it returns the first active DECB.

### ANY

Returns the next allocated DECB without first verifying that the DECB is active. The register specified by the DECB parameter indicates the address of the current DECB. If the register specified by the DECB parameter contains a value of zero, the first DECB will be returned without verifying the active state of the DECB.

### WKREG=(*reg*)

Specifies an available general register (R1–R7, R14, or R15) to be used on the macro call. If no register is specified, the default register is R14.

### LEVEL=*level*

Specifies a valid ECB data level (D0–DF).

### ERROR=*label3*

The label in which to branch if an error occurs during processing of the macro. If ERROR is not specified and an error occurs, control will return to the next sequential instruction (NSI). The caller then checks the contents of the register specified on the RC parameter for the associated return code.

### RC=(*reg*)

Specifies the general register (R0–R7, R14, or R15) where the return code is placed when the macro ends. For E-type programs, R15 must be specified.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- When called from an E-type program, the contents of R14 and R15 cannot be predicted. The contents of all other registers are maintained.

- The register specified on the RC parameter contains the return code from one of the following:

**DECBC\_OK**

The request completed successfully.

**DECBC\_DUPNAME**

The NAME parameter specified for FUNC=CREATE contains a DECB name that already exists for this ECB.

**DECBC\_NOTFOUND**

A DECB matching the input search criteria could not be found when specifying FUNC=LOCATE.

**DECBC\_NOTVALID**

The DECB specified for FUNC=VALIDATE did not reference a valid DECB address or the DECB is no longer in use.

## Programming Considerations

- The DECB is an alternative to standard ECB data level information. Data level information is used to specify information about I/O request (core block reference word (CBRW) and file address reference word (FARW)) fields. The DECB fields specify the same CBRW and FARW information without requiring the use of a data level in the ECB. All the same requirements and conditions that apply to the CBRW and FARW in the ECB also pertain to the same field information in the DECB.
- The FARW in the DECB provides storage for an 8-byte file address. The DECB layout is in the IDECB DSECT (see the IDECB DSECT for more information about the fields in a DECB).
- FUNC=SWAPBLK is not intended to perform the same operation as the FLIPC macro, which is for ECB data levels. When FUNC=SWAPBLK is specified, only the CBRW information is exchanged between the specified DECB and ECB data level.
- Macros that support the use of a DECB (such as FILEC, FINDC, and others) will only accept a DECB address as a valid reference to the DECB. If an application does not maintain the address of a particular DECB and instead maintains the name of the DECB, the caller will first have to issue the DECB macro with the FUNC=LOCATE parameter to obtain the address of the desired DECB. The resulting address of the DECB can then be passed to the subsequent macro call.
- FUNC=LOCATE specified with the CHAIN parameter is intended for use by system utilities that have a requirement to step through each DECB associated with a particular ECB. This parameter must only be used when the application is running in the ECB virtual memory (EVM).
- The letters I, i, TPF, TPF\_, tpf, and tpf\_ are reserved for future use by IBM for DECB name spaces.

## Examples

- The following example obtains a DECB and assigns it a name of TESTDECB. The address of the newly created DECB is returned in register 7 (R7).

```
DECBC FUNC=CREATE,DECB=(R7),NAME=NEWDECB
```

```
NEWDECB DC CL16'TESTDECB'
```

- The following example exchanges the core block information from the DECB whose address is represented in R5 and data level 3 of the ECB.

```
DECBC FUNC=SWAPBLK,DECB=(R5),LEVEL=D3
```

## DECBC

- The following example releases the DECB named DATABLK1.

```
DECBC FUNC=RELEASE,NAME=RELDECB
```

```
RELDECB DC CL16'DATABLK1'
```

- The following example locates the DECB named DATABLK2 and returns its address in R3.

```
LA R1,DECBNAME
```

```
DECBC FUNC=LOCATE,DECB=(R3),NAME=(R1)
```

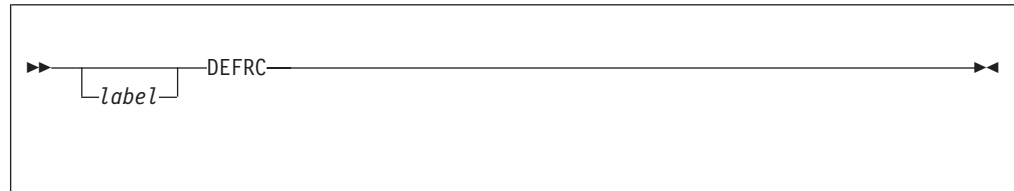
```
DECBNAME DC CL16'DATABLK2'
```

## DEFRC–Defer Processing of Current Entry

This general macro is used to defer processing of an entry, until the amount of activity in the system is sufficiently low to allow for completion of this low priority task.

The entry is placed on the defer list, the lowest priority CPU loop list, but continues to hold all core blocks assigned.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The ECB must not be holding any file records, such as FINHC, FIWHC, or others.

### 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 condition code is not saved across this macro call.

### Programming Considerations

- This macro can be executed on any I-stream.
- Following execution of this macro, the ECB is added to the Defer list. Accordingly, control can be transferred for processing of another entry.
- The execution of this macro will reset the 500-millisecond program timeout.
- The macro trace collection for DEFRC will compress multiple occurrences of DEFRC into two entries. This prevents programs that issue successive DEFRCs from filling up the macro trace table by keeping only the first and last DEFRC trace information.

In addition to the normal macro trace information the first DEFRC trace entry contains a count of suppressed entries.

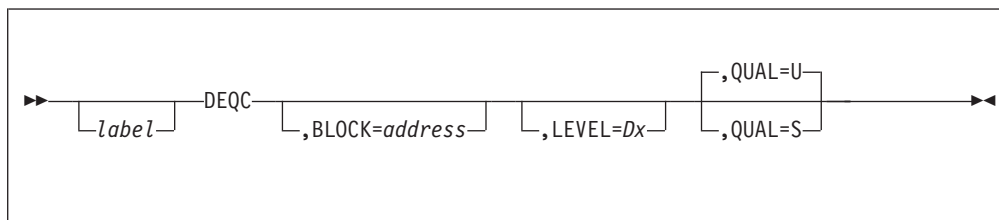
### Examples

None.

## DEQC–Dequeue from Resource

Use this general macro to inform the control program that the entry control block (ECB) has finished with a shared resource. The DEQC macro is used with the ENQC or SANQC macro.

### 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 on which the ECB was enqueued using the ENQC or SANQC macro. 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 on which the ECB was enqueued using the ENQC or SANQC macro. BLOCK and LEVEL are mutually exclusive.

#### **QUAL**

Subsystem qualification for the resource. The default value is U.

- U** Subsystem qualification applies. The resource name is subsystem unique and is qualified by the Data Base 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 or ENQC 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 coded, two different resources are assumed to exist. The SANQC or ENQC macro and its corresponding DEQC macro must have the same QUAL value coded.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
  - DEQC using LEVEL

The symbolic name of the shared resource, 8 characters in length, must be in CE1FAx, where x is the specified level.

- DEQC using BLOCK

The symbolic name of the shared resource, 8 characters in length, must be in the 8-byte area pointed to by the BLOCK parameter.

### Return Conditions

- Control is returned to the next sequential instruction.

- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.
- DEQC of a level holding a block will result in a system error and the ECB will be exited.
- Attempting to DEQC from a resource the ECB is not holding will result in a system error and the ECB will be exited.

## **Programming Considerations**

- This macro can be executed on any I-stream.
- The resource specified with this macro must be a resource that was previously defined by an ENQC or SANQC macro.
- See *TPF System Macros* for more information about the SANQC macro.

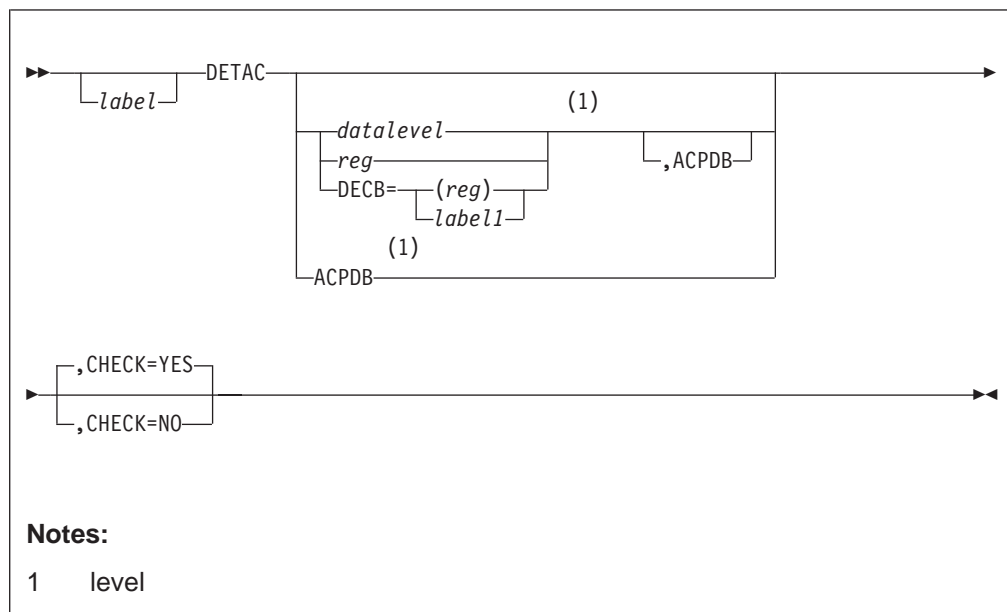
## **Examples**

None.

## DETAC—Detach an ECB Working Storage Block

Use this general macro to detach a storage block from the specified entry control block (ECB) data level or data event control block (DECB) while allowing the ECB data level or DECB to be reused. The block is saved and can be reattached by using the ATTAC macro.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### *level*

The ECB data level from which the block is to be detached. Valid formats are:

#### *datalevel*

Valid values are D0–DF.

#### *reg*

A register containing the ECB data level value.

#### DECB=(*reg*)|*label1*

The label or general register (R1–R7) containing the address of the DECB where the block will be detached.

#### ACPDB

The user is TPFDF. The ECB data level value is in register 7 (R7) if *level* or DECB is not specified.

#### CHECK

Specify one of the following:

#### YES

If specified and the ECB data level or DECB is not holding a block, the service routine will issue a system error.

#### NO

The macro service routine will make the ECB data level or DECB reusable without validating that a block is held.



## Entry Requirements

- R9 must contain the address of the ECB being processed.
- When CHECK=YES is specified, the ECB data level or DECB must be holding a storage block.
- When ACPDB is specified, a unique identifier must be supplied in the file address reference word (FARW) of the indicated ECB data level or DECB. This identifier will be used when reattaching the block and allows the order of reattachment to be controlled by the user.

## 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 the ECB data level is coded as a register and that register is not R7, the contents of the specified register will be loaded into R7.
- The specified ECB data level or DECB will no longer be holding a storage block.

## Programming Considerations

- This macro can be run on any I-stream.
- The specified ECB data level or DECB must be holding a block at the time the DETAC macro is issued unless CHECK=NO is specified.
- Excessive use of the macro can cause working storage to be depleted; therefore, carefully monitor its use.
- The ACPDB option affects how detached blocks are reattached. Refer to the ATTAC macro.
- The number of DETACs issued is kept in the ECB. The count is maintained on an ECB data level or DECB basis. Only 255 DETACs are allowed to be issued against a specific ECB data level. However, more than 255 DETACs are allowed to be issued against a DECB (the count is maintained in field IDECDET of the DECB). For the ACPDB option, the count is maintained on an ECB basis and will only allow 255 ACPDB DETACs to be issued. When the DETAC count is exceeded, the service routine will issue a system error.

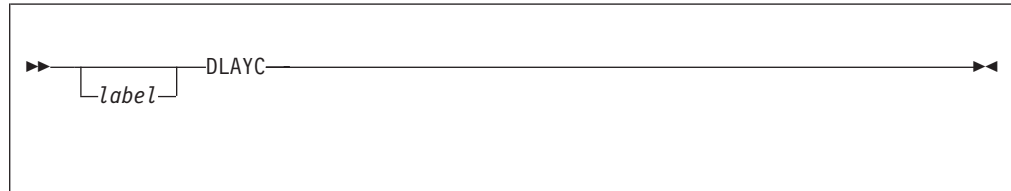
## Examples

None.

## DLAYC–Delay Processing

This general macro is used to delay the processing of an entry and allow processing of other entries. The entry is placed on the CPU loop input list.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### Entry Requirements

R9 must contain the address of the entry control block (ECB) being processed.

### 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 condition code is not saved across this macro call.

### Programming Considerations

- This macro can be run on any I-stream.
- After this macro runs, the ECB is added to the CPU input list. Accordingly, control may be transferred for processing of another entry.
- ***Use of this macro should be limited, because excessive use of this macro will cause a low-core condition in the real-time system.***
- Records should not be held by the ECB when the delay is issued. If the entry issuing the DLAYC is holding a record, a number of entries (also holding storage blocks) could be queued for that record. This could cause a lockout condition, in which there is not enough storage to service the input list containing the delayed entry.
- When this macro runs, the 500-millisecond program timeout is reset.
- The macro trace collection for DLAYC compresses multiple occurrences of DLAYC into two entries. This prevents programs issuing successive DLAYCs from filling up the macro trace table by keeping only the first and last DLAYC trace information. In addition to the normal macro trace information the first DLAYC trace entry contains a count of suppressed entries.

### Examples

None.

---

## DPANL—Terminal Panel Display

This general macro instruction, in conjunction with the DDATA macro instruction (DDATA), is used in application development to declare a data structure used in processing terminal displays. DPANL and DDATA macro instructions (called map statements) written by an application programmer must be placed on a map source statement library with a utility program. Map statements are then included in ECB-controlled source code through the assembly language COPY instruction. The copied map statements have the effect of declaring symbolic names used in a display data structure, because the DPANL and DDATA instructions expand into a DSECT included in the source code. The DSECT is adjusted and a register is assigned as the result of parameters included in the map statement coding.

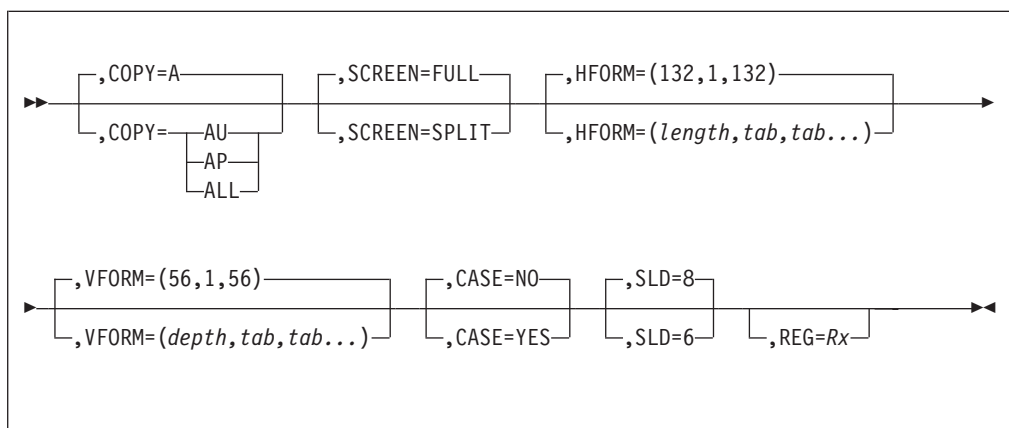
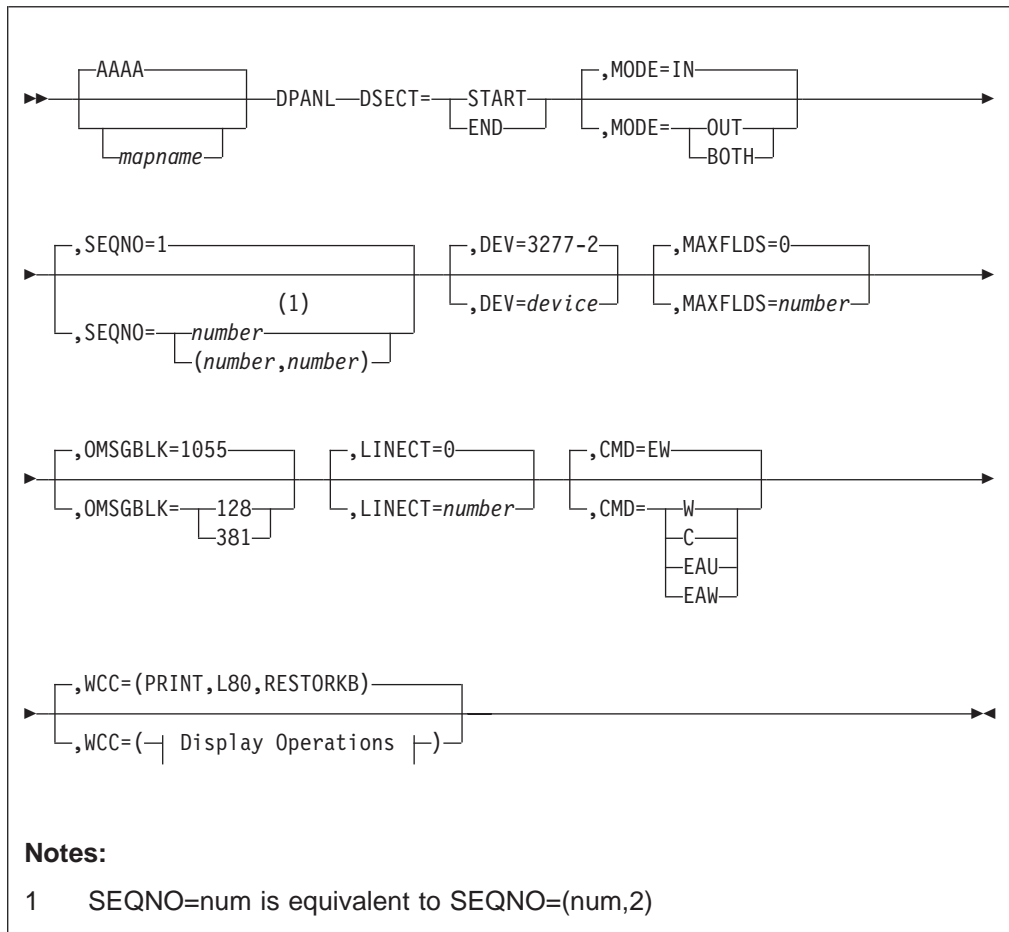
The effect of using map statements is analogous to the use of a data macro instruction. However, map statements do not appear in the source code of ECB-controlled programs. This means that source code does not contain the register assignment of the DSECT generated as the result of copied map statements.

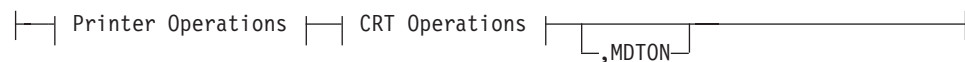
A map statement on the map source statement library is used in two different ways:

- A map statement is copied into ECB source code as outlined previously. Specifically, the DPANL macro instruction is used to define the register assignment, the maximum displacement of the generated DSECT, the map sequence number equate, and the fields comprising the standard header.
- A map statement is used by a portion of the mapping support package to create a physical terminal map record (AS0MP) in the fixed file area. These records are used by the system-supplied ECB-controlled programs to superimpose or delete control characters in data streams processed by application programs. Specifically, the DPANL macro instruction is used to define terminal and screen characteristics that govern the presentation of an entire display.

These DPANL macros come after the DDATA macro field definitions.

## DPANL Format



**Display Operations:****Printer Operations:****CRT Operations:****mapname**

A unique 1-to-4-character name (alpha, numeric, special character) must be assigned by the user to this format definition. This field must begin in column 1 of the card image. The first character must be alpha. It is recommended that this same name be used to install the complete set of map statements on the Map Source Library. When the mapname is omitted, the name 'AAAA' is assumed by default.

**DPANL**

The macro statement name must follow the symbolic map name. There must be at least one blank column between it and the map name. The designation must appear as DPANL.

**DSECT=START**

This parameter establishes the start of the map source statements that define this specific format; it must be coded as shown in the example.

**MODE=IN|OUT|BOTH**

This optional keyword allows the user to specify whether the symbolic reference module (DSECT) and the physical map record (MSP) are to be formatted by the Mapping Support Package for input data streams (MODE=IN) or output data streams (MODE=OUT). When the user defines a single map that is to be used for input and output data streams, MODE=BOTH is coded. This will cause the generation of 2 DSECTs, one formatted as input and the other as output, in the user program. Similarly, 2 Terminal Map Records will be formatted for this specific map.

When this keyword is omitted, the mode is assumed to be IN.

When a single map is defined by the user for both input and output, (that is, MODE=BOTH), two COPY 'mapname' statements must be inserted in the

source program. The two statements are identical. The purpose of this is to provide for the subsequent generation of the input and the output related symbolic DSECTs.

**SEQNO=***number*[(*number*,*number*)]

This keyword allows the user to specify a unique map sequence number for this format. When MODE=IN or MODE=OUT, only 1 number is specified. When MODE=BOTH, the user must code 2 different numbers. The numbers assigned to a particular map must not be duplicated or have been assigned to other maps.

This parameter is optional; the range of valid numbers is 1–9999. When SEQNO is omitted, SEQNO=1 is assumed.

**Note:** When 2 numbers are coded, they must be enclosed in parentheses.

**DEV=***device*

Check devices specified in DEV explanation

This optional keyword allows the user to specify the terminal types associated with this particular map format. When MODE=IN or MODE=BOTH is coded, the only valid terminal types that may be specified are 3277-1, 3277-2, 3278-1,2,3,4 default size, and 3278-1,2 alternate size. When MODE=OUT, any combination of terminal types, as shown in the example, may be coded for the DEV= keyword. When this keyword is omitted from the map definition, DEV=3277-2 is the assumed device type.

For input maps, the following devices are supported:

| DEVICE TYPE    | DEVICE CODE TO BE USED |
|----------------|------------------------|
| 3277 MOD 1     | 3277-1                 |
| 3277 MOD 2     | 3277-2                 |
| 3278 MOD 1 Def | 3278-1                 |
| 3278 MOD 2 Def | 3278-2                 |
| 3278 MOD 3 Def | 3278-3                 |
| 3278 MOD 4 Def | 3278-4                 |
| 3278 MOD 1 Alt | 3278-1A                |
| 3278 MOD 2 Alt | 3278-2A                |

For output map generation, the following devices are supported:

| DEVICE TYPE    | DEVICE CODE TO BE USED |
|----------------|------------------------|
| 3277 MOD 1     | 3277-1                 |
| 3277 MOD 2     | 3277-2                 |
| 3284 MOD 1     | 3284-1                 |
| 3284 MOD 2     | 3284-2                 |
| 3286 MOD 1     | 3286-1                 |
| 3286 MOD 2     | 3286-2                 |
| 1977           | 1977                   |
| 1977 with tab  | 1977-T                 |
| 1980 MOD 21    | 1980-1                 |
| 1980 MOD 24    | 1980-4                 |
| 3278 MOD 1 Def | 3278-1                 |
| 3278 MOD 2 Def | 3278-2                 |
| 3278 MOD 3 Def | 3278-3                 |
| 3278 MOD 4 Def | 3278-4                 |
| 3278 MOD 1 Alt | 3278-1A                |
| 3278 MOD 2 Alt | 3278-2A                |
| 3287 MOD 1 Def | 3287-1                 |
| 3287 MOD 2 Def | 3287-2                 |
| 3287 MOD 3 Def | 3287-3                 |
| 3287 MOD 4 Def | 3287-4                 |

```

3287 MOD 1 A1t 3287-1A
3287 MOD 2 A1t 3287-2A
3287 MOD 1 SLU 3287-1S
3287 MOD 2 SLU 3287-2S

```

Although the previous device parameters are used to create the map records, they will be checked during the assembly process of the associated symbolic reference module to ensure correct coding. Users with 3289 printers should code the 3287 equivalent types. Users with 3288 printers should code the 3286 equivalent types.

**MAXFLDS=number**

This optional keyword is valid only when MODE=IN or MODE=BOTH has been specified. This parameter provides a count of the maximum number of fields that may be present in an input data stream. This specification will provide for the allocation of an area in the user's prime DSECT block, which enables the user program to interpret the presence or lack of a data field. There will be 1 bit assigned for each input field. The relative position of each bit indicator in the IMF field of the user block corresponds to a single associated input data field in the DSECT. When the bit=ON, the field has been entered; when the bit=OFF, the field has not been entered. For fields defined as REPEAT (see DDATA description), the count must include the number of input fields created for that single DDATA statement.

When this keyword is absent from the DPANL statement, the assumed count is set to zero, no area will be allocated in the user DSECT, and no bit strings will be generated by the Mapping Package in the user block.

**OMSGBLK=128|381|1055**

This optional keyword, is valid on maps defined as MODE=OUT, MODE=IN, or MODE=BOTH. This parameter defines the maximum size of the user output block and the size of the user input block, and therefore sets the maximum displacement of the last field that can be contained in the DSECT. When this keyword is omitted from the DPANL statement, OMSGBLK=1055 is assumed.

**Note:** When the defined block size is exceeded, an ORG is generated to reset the displacement of the next field back to the beginning of the DSECT.

**LINECT=number**

This optional keyword is valid only for the 1980/24 and the 1977 hardcopy terminal types. When this parameter is provided on the DPANL statement, it specifies the number of lines on a particular form. The default is LINECT=0.

The following 4 keywords are only for output maps and are related to the IBM 3270 Information Display System terminal types (including 3278 and 3287 terminals). For additional information concerning CMD, WCC, and COPY, see the IBM 3270 Information Display System Component Description Manual. These keywords are valid when MODE=OUT or MODE=BOTH is specified.

**CMD=W|EW|C|EAU|EWA**

This optional keyword lets the user indicate specific 3270 operations. Utilized with the WCC and COPY keywords, the user may initiate the writing and erasing of data in a selected 3270 device buffer. When CMD=W, a Write operation character is generated; CMD=EW, an erase buffer and write operation character, is generated; CMD=C, a copy control command character is generated; and CMD=EAU, a command to erase all unprotected characters is generated. This last command clears all unprotected buffer character locations to nulls, resets to 0 the MDT bit for each unprotected field, unlocks the keyboard, resets the AID byte, and repositions the cursor to the first character location in the first unprotected field of the buffer. The CMD=EWA (erase/write

## DPANL

alternate) parameter is valid only for 3278-1 and 3278-2 terminals. The EWA command sets the buffer of the terminal to its alternate size. If this Keyword is omitted, CMD=EW is assumed.

### **WCC=PRINT,L40|L64|L80|NL,ALARM,RESTORKB|LOCKKB,MDTON**

This optional keyword is used with the 3270 command (CMD) operand.

For printer operations, the user can specify the printout format and the initiation of the printout operation at the completion of the Write operation.

For CRT display operations, the user may specify that the alarm is sounded and the keyboard be restored or both. For either printer or CRT operations, the user may also indicate that all MDT bits in the existing buffer data of the referenced devices be reset before the Write operation.

When WCC=PRINT, the user must indicate an associated line length specification; the assumed default for this parameter is the 80-character print line (L80). When WCC= is not present on the DPANL statement, the keyboard restore parameter (RESTORKB) is assumed (the default WCC for a printer will be PRINT,L80). The parameters, when coded, must be in the order that appears at the beginning of this section.

### **COPY**

This optional keyword is valid only when CMD=C has been specified. This parameter denotes the type of data to be transferred from the buffer of 1 device to another attached to the same 3271 control device.

**A** Only attribute characters are transferred.

#### **AU**

Attribute characters and unprotected alphameric fields are copied.

#### **AP**

All attribute characters and protected alphanumeric fields are copied.

#### **ALL**

The entire buffer contents are copied.

When the COPY= keyword is not coded, but CMD=C has been, the 'A' parameter is assumed.

### **SCREEN=FULL|SPLIT**

This optional keyword can be used to indicate screen splitting. When the user wants to obtain split screen capabilities, that is, the separate paging and scrolling of two distinct portions of a single display, SCREEN=SPLIT is specified. When no split screen capability is sought, the user either specifies SCREEN=FULL or omits the operand entirely. If this keyword is omitted, SCREEN=FULL is assumed.

The following 3 commands are related only to the IBM 3278/3787 information display system terminal types. The commands are only valid for the 3287 printers. When any of these commands are coded, and the device is a 3287, the implication is that the data stream to be formatted is LU type 1 with SCS (SNA character string) codes. When these commands are not coded and the device is a 3287, the data stream is formatted as LU type 3; that is, as a 3270 printer data stream whose buffer addresses depend on whether the default or alternate size is used.

### **HFORM=length,tab2,tab...**

This optional keyword can be used to define the horizontal format including left and right margins and horizontal tab stops. The first parameter must define the line length. The default values for the HFORM operand are: line length=132,



first tab=1, last tab=132 (for example, HFORM=(121,1,10,15,100), means the line length is 121, tab settings are at positions 1, 10, 15, 100).

**VFORM=depth,tab,tab...**

This optional keyword can be used to define the vertical format including the maximum presentation line, top and bottom margins, and vertical tab stops. The first parameter must define the form depth. The default values for the VFORM operand are: form depth=56 lines, first tab=1, last tab=56 (for example, VFORM=(56,1,56)).

**CASE=YES|NO**

This optional keyword can be used to convert all input data to upper case. It is not valid for output map definitions. CASE=NO is the default. CASE=YES will cause conversion.

**SLD=6|8**

This optional keyword can be used to specify the distance to be moved for a single line vertical spacing. This Set Line Density (SLD) keyword is valid only for the printers supporting the LU type 1 data stream. Valid parameters are 6 (6 lines/inch) or 8 (8 lines/inch). The default is 8.

The concluding DPANL statement appears as follows:

```
DPANL DSECT=END,REG=Rx
```

**DPANL**

The macro statement name must be the first data on the card; no data must appear to the left of these 5 characters.

**DSECT=END**

This required parameter establishes the conclusion of a series of map source statements that defines a specific format (map); it must be coded in the following manner: DSECT=END.

**REG=Rx**

This optional keyword can be used to establish the base register for this specific map. The user specifies the general register to be associated with the defined map. However, this keyword does not load the register specified with an address; it merely establishes the symbolic addressability of each field contained in the assembled DSECT. When this keyword is omitted, no base register is established for the map.

## Entry Requirements

All DPANL statements along with their associated DDATA statements will be installed on a source library. For reference, the user must issue a COPY statement in the source program text.

## Return Conditions

Return conditions are not applicable. DPANL provides a DSECT only.

## Programming Considerations

- When the DPANL statements are coded to describe a display panel, a DSECT=END must be the last statement. For assembly purposes, all statements will be installed in a source library and accessed by a COPY statement.
- Checks will be made to determine proper sequence of the DPANL statements. Also, all parameters will be checked for proper coded entries. If no entries are made, default parameters will be taken when applicable.

## DPANL

- The coded statements are to be installed on a source library in order to be accessed by a COPY statement. It is recommended that the name used to install the map statements on the Map Source Library be the same as the map name used on the initial DPANL statement of a map definition.
- I-stream restrictions are not applicable to the DPANL macro.

## Examples

- Input module called menu:

```
* STATEMENTS FOR INPUT MAP

MENU DPANL DSECT=START,MODE=IN,SEQNO=1,MAXFLDS=3
 DDATA
 DPANL DSECT=END,REG=R2C
```

- Output module called menu:

```
MENU DPANL DSECT=START,MODE=OUT,WCC=(ALARM,RESTORKB),
 SEQNO=2
 DDATA
 DPANL DSECT=END,REG=2
```

- Input and output module called menu:

```
MENU DPANL DSECT=START,MODE=BOTH,SEQNO=(3,4),MAXFLDS=17

* STATEMENTS FOR INPUT MAP

MENU DPANL DSECT=START,MODE=BOTH,SEQNO=(3,4),MAXFLDS=17
 DDATA
 DPANL DSECT=END,REG=R4

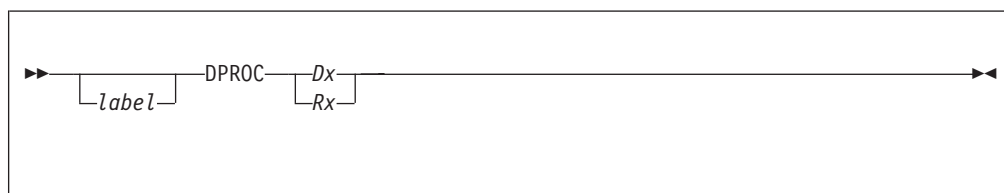
* STATEMENTS FOR OUTPUT MAP

MENU DPANL DSECT=START,MODE=BOTH,SEQNO=(3,4),MAXFLDS=17
 DDATA
 DPANL DSECT=END,REG=R4
 DDATA
 DPANL DSECT=END,REG=R4
```

## DPROC—Program Record Determination

This general macro provides a centralized mechanism for interrogation of a DASD address to determine if the record is a program record.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* The file address reference word on level *Dx* must contain the address to be interrogated.

*Rx* The register specified contains the input file address to be interrogated. The valid registers are R0–R7.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- **When the condition code is 0.**  
The address specified is a program record.
- **When the condition code is 1.**  
The specified address is not a program record address.
- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of registers R0 thru R7 are preserved across this macro call.

### Programming Considerations

- The ECB reference register (R9) must contain the address of the ECB being processed.
- The DPROC macro may be used on any I-stream.

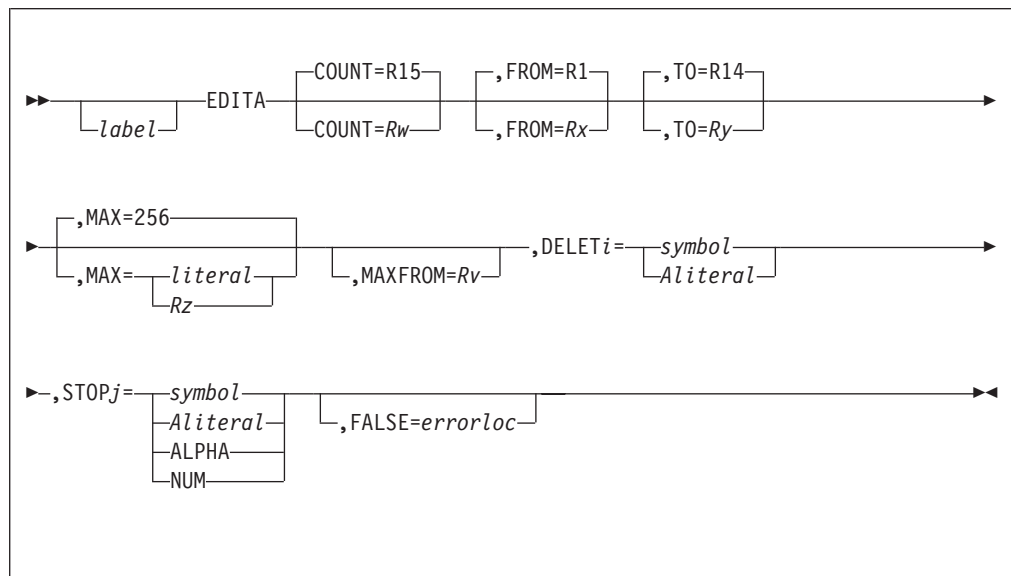
### Examples

None.

## EDITA—Edit and Move Data

Use this general macro to edit and move data. This macro is used to scan a string of characters for a delimiter. It deletes certain characters in the string (such as blanks) and moves the significant characters in the string to a prescribed work area, keeping count of the number of characters moved. The operation ends when a specified character (such as an end-of-message character) is encountered or when a specified number of characters have been moved. This is a data-manipulating type of macro. The characters are edited from left to right and stored left-justified in the destination area.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **COUNT=Rw|R15**

Significant character count. It designates a symbolic general register, other than R0, which will contain the count of significant characters after execution of the macro. If this parameter is omitted, R15 is assumed for use.

#### **FROM=Rx|R1**

This is the source location register. It designates a symbolic general register, other than R0, which contains the address of the first character in the string to be edited. If this parameter is omitted, R1 is assumed to contain that address.

#### **TO=Ry|R14**

This is the destination location register. It designates a symbolic general register which contains the address of the first position into which the significant characters are to be moved. R14 will be assumed if this keyword is omitted.

#### **MAX=literal|256|Rz**

Specify one of the following:

*literal*

It defines the maximum number of significant characters to be moved. This parameter will end the operation if the specified delimiter is not encountered, where  $1 \leq \text{literal} \leq 256$ . If this parameter is omitted, the maximum number of characters moved will be 256 characters.

*Rz* This is any symbolic general register that contains a binary count defining the maximum number of significant characters to be moved. The value contained in the register must conform to the limits set up for literal usage.

#### **MAXFROM=*Rv***

This is any symbolic general register other than R0 that contains the maximum, or last, address of the source location to be edited. Editing ends when the address of the character being scanned (in the source location) exceeds MAXFROM before a stop delimiter has been found and before the maximum number of characters has been moved. The default is to not check for the maximum source location address as an ending condition.

#### **DELET*i***

Where *i* is 1-3. These parameters identify as many as 3 characters that, if found at the source location (specified by the FROM parameter), will not be moved to the destination location. These parameters may be defined as symbols or as literals. The literal may be any self-defining expression that can be coded in the immediate field of a CLI instruction.

These parameters must be used in sequence. That is, if DELET2 is used, DELET1 must be included, and if DELET3 is used, both DELET1 and DELET2 must be included.

##### *symbol*

The symbolic location of a byte that will be used for comparison. The symbol cannot begin with the letter A.

##### *Aliteral*

The letter A followed by any self-defining expression that could be coded in the immediate field of a CLI instruction as given in the following examples.

Examples of legal delimiters:

| <b>Delimiter</b> | <b>Binary Representation</b> |
|------------------|------------------------------|
| AX'2F'           | 00101111                     |
| AC' '            | 01000000                     |
| A#CAR            | 00010101                     |

Examples of illegal delimiters:

| <b>Delimiter</b> | <b>Explanation</b>        |
|------------------|---------------------------|
| X'2F'            | Not preceded by an A.     |
| AX'CFA3'         | More than 1 byte defined. |
| C'A'             | Not preceded by an A.     |
| AC'AB'           | More than 1 byte defined. |

#### **STOP*j***

Where *j* is 1-3. These parameters specify characters which cause EDITA to cease editing. They can be specified as: *symbol*, *Aliteral*, ALPHA or NUM.

If STOP1 is omitted an end-of-message-complete character (#EOMC) will be assumed as the option. If STOP2 or STOP3 is omitted no option will be assumed. If STOP2 is omitted STOP3 will not be examined for options.

Examples of legal STOP combinations are:

```
EDITA STOP1
EDITA STOP1,STOP2,STOP3
EDITA STOP2 /* STOP1 assumed EOMC */
```

## EDITA

Illegal combinations are:

|       |             |                  |    |
|-------|-------------|------------------|----|
| EDITA | STOP1,STOP3 | /* STOP2 missing | */ |
| EDITA | STOP3       | /* STOP2 missing | */ |

### *symbol*

This is the symbolic location of one, two or three bytes that will be used for comparison to stop the EDITA scan. For example: EDITA STOP1=PCA412 is coded and the symbolic location, PCA412 defines a C'/. This will cause editing to cease when a slash (/) is found in the FROM field. The symbol cannot begin with the letter A.

### *Aliteral*

The letter A followed by any self-defining expression that can be coded in the immediate field of a CLI instruction.

### **ALPHA**

Editing is stopped at the first nonnumeric character (not a digit, 0–9, X'F0'–X'F9').

### **NUM**

Editing is stopped at the first numeric character (digit, 0–9, X'F0'–X'F9').

Examples of legal delimiters:

| Delimiter | Binary Representation |
|-----------|-----------------------|
| AX'2F'    | 00101111              |
| AC' '     | 01000000              |
| A#EOM     | 01001110              |
| ALPHA     | 11110000 (BNO)        |
| NUM       | 11110000 (BO)         |

Examples of illegal delimiters:

| Delimiter | Explanation               |
|-----------|---------------------------|
| X'2F'     | Not preceded by an A.     |
| AX'CFA2'  | More than 1 byte defined. |
| C' '      | Not preceded by an A.     |
| AC'@@@'   | More than 1 byte defined. |

### **FALSE=errorloc**

This the symbolic location where control is to be transferred in the event that the specified delimiter is not found; that is, if the maximum number of characters has been scanned or moved. If this parameter is omitted, return will be to the next sequential instruction with no other indication of the error. The user could determine this, however, because the source location register would not be pointing to the specified delimiter.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The source location register (R1 is default) must contain the address of the first character to be edited.
- The destination location register (R14 is default) must contain the address of the first position into which the significant characters are to be moved.

- The significant character count register (R15 is default) must be free for use. It is cleared by the macro and its contents are not saved.

## Return Conditions

- When the maximum number of characters has been moved (MAX parameter) or the scan exceeds the maximum source location address (MAXFROM parameter), control is returned to either the location specified by FALSE parameter or to the next sequential instruction (if the FALSE parameter is not used).
- The source location register (specified by the FROM parameter) will contain the address of the specified stop delimiter in the character stream. The delimiter is moved. If the delimiter is not found, the register will contain the address of that character position, which is equivalent to the first character position plus the maximum number of significant characters scanned.
- The destination location register (specified by the TO parameter) will contain the address into which the last character was moved if a stop character was encountered; it will contain the address +1 if the maximum number of significant characters was moved (MAX parameter) or the scan exceeds the maximum source location address (MAXFROM parameter).
- The significant character count register (specified by the COUNT parameter) will contain the number of significant characters moved (not including the stop character).
- R1, R14, and R15 are the default registers used. However, the programmer can explicitly state other registers when necessary.
- The condition code will be changed as a result of execution of this macro.

## Programming Considerations

- This macro can be executed on any I-stream.
- If a FALSE= parameter is used, and MAX or MAXFROM have occurred before a valid STOP parameter has been found, control will be returned to the location specified by FALSE. Removing the FALSE parameter eliminates the error checking. Control is returned to the next sequential instruction.  
(Coding the FALSE= parameter does not require any additional storage use.)
- The matching STOPn parameter **will** be moved to the destination field.
- The contents of the MAX= register are **not** validated. This could cause extended or erroneous processing if an incorrect value is used.
- Invalid *symbol/literal* definitions may result in assembly errors.
- MAX= is compared versus the source location (FROM=).
- MAXFROM= is compared versus the source location (FROM=).
- COUNT= is computed versus the destination location (TO=).
- If MAX= is defined as a literal, a halfword literal will be defined. An LTOrg statement should be part of the program.
- There is **no** cross parameter validation for register conflicts.

## Examples

The following is an example taken from segment WGA1:

```
* EDIT THE INPUT MESSAGE TO REMOVE ALL BLANKS AND CARRIAGE RETS
 LA R14,MIOACC
 LR R1,R14
 EDITA DELET1=AX'40',DELET2=A#CAR,FALSE=WGA1EB
* UPDATE ITEM LENGTH COUNTER TO REFLECT EDIT
 LA R15,4(R15)
```

## EDITA

```
 STH R15,MIOCCCT
* IS THE INPUT MESSAGE THE PROPER LENGTH ?
 .
 .
 .
WGA1EB MVI EBW044,X'01' INVLD FORMAT
```

In the previous example, CAR refers to the symbol for carriage return.

The following example is taken from segment UIM1:

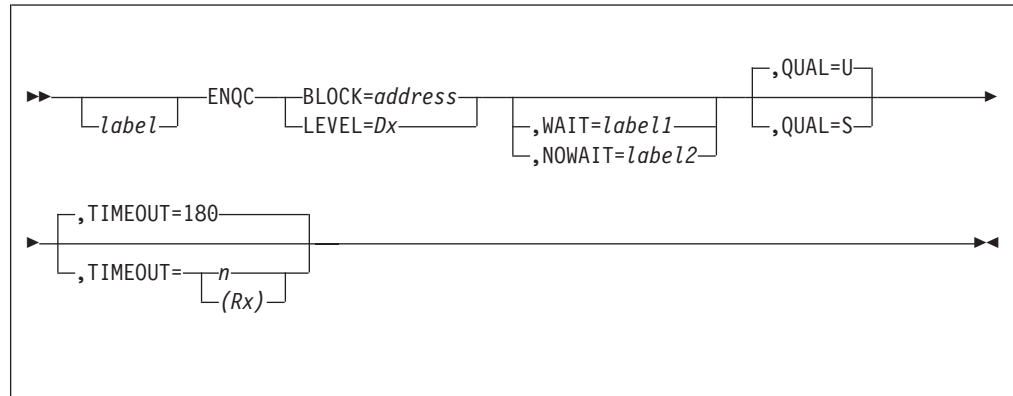
```
UIM1NZ EQU *
 LA R0,MIOLN0 R0=ADDRESS FIRST COUNTED BYTE
 LH R1,MIOCCCT R1=MESSAGE COUNT
 AR R1,R0 R1=ADDRESS OF EOM+1
 LA R0,MIOACC R0=ADDRESS OF FIRST CHARACTER TEXT
 SR R1,R0 R1=COUNT OF CHARACTERS TO BE SCANNED
 LA R2,MIOACC R2=FROM LOCATION
 LR R7,R2
 EDITA FROM=R2,T0=R7,DELET1=AC' ',DELET2=A#CAR,MAX=R1
 LA R15,4(R15) ADJUST COUNT
 STH R15,MIOCCCT STORE COUNT
```



## ENQC—Define and Enqueue Resource

This general macro is used to define to the control program a shared resource and to control access to the resource among ECBs. This macro is used with the DEQC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**BLOCK=address**

If specified, it is the address of an 8-byte area which contains the ENQC resource name. BLOCK and LEVEL are mutually exclusive.

**LEVEL=Dx**

If specified, it is a file address reference word (D0-DF) which contains the name of resource. BLOCK and LEVEL are mutually exclusive.

**WAIT=label1**

The label of an instruction to be given control if the ECB waited before gaining control of the resource. Otherwise, the next sequential instruction will be given control on return from the execution of the macro. WAIT and NOWAIT parameters 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 will be passed to the specified label without the ECB giving up control. NOWAIT and WAIT parameters are mutually exclusive.

**QUAL**

Subsystem qualification for the resource. The default value is U.

**U** Subsystem qualification applies. The resource name is subsystem unique and is qualified by the Data Base Index (DBI) value for the subsystem.

**S** System-wide qualification applies. The resource name is not subsystem unique. Any ECB issuing an ENQC with this system name and QUAL=S will be enqueued on the same named resource. If two ENQCs are issued with the same resource name but different QUAL values, then two different resource names are assumed to exist. The DEQC must have the same QUAL value as the ENQC.

## ENQC

### **TIMEOUT=n|(Rx)**

The number of seconds that the ECB expects to hold the resource. The value may be specified by a self-defining value (from 0 to 4096) or by a value (from 0 to 32 768) in a register.

The timeout does not start until there is another ECB waiting. If the time elapses, the ECB will be exited with a system error. A default value of 180 seconds is used if none is specified. If 0 is specified, the timeout function is turned off.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- When you specify the LEVEL parameter, the symbolic name of the shared resource, 8 characters in length, must be in CE1FAx, where x is the specified level.
- When you specify the BLOCK parameter, the symbolic name of the shared resource, 8 characters in length, must be in the 8-byte area pointed to by the BLOCK parameter.

## Return Conditions

- If neither WAIT or NOWAIT is specified control is returned to the next sequential instruction when the ECB owns the resource. If the resource is already in use and NOWAIT is specified, control is given to the specified label without loss of control for the ECB, but the ECB will not own the resource. If the resource was already in use and WAIT was specified, control is given to the specified label when the ECB owns 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 be executed on any I-stream.
- If LEVEL is used, it must not be holding a block. If it is, a system error is executed and the ECB is exited.
- When finished with the resource, the ECB must issue a DEQC macro. If the ECB exits holding a resource, a system error is issued and the resource is freed.

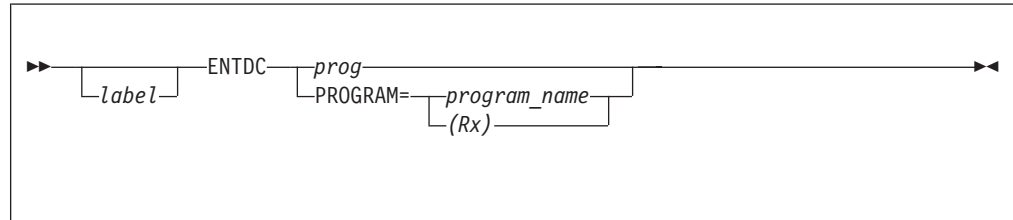
## Examples

None.

## ENTDC—Enter a Program and Drop Previous Programs

This general macro transfers control to the specified operational program and releases all program records held by the entry control block (ECB).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be entered. This method generates a VCON resolved at link edit time into a Program Allocation Table (PAT) displacement. This is the preferred method for specifying the program name.

#### PROGRAM

The name of the program can alternately be provided using the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the PAT displacement. This method has a longer path length than the one described above.

*program\_name*

The name of the program that is to be entered.

*(Rx)*

A register (R0–R7) that contains the address of the program name.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

Control is never returned to a program that issues the ENTDC macro.

### Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB being processed before using this macro.
- Using the PROGRAM parameter sacrifices some performance.
- Enter is a system critical path. Adding even 1 instruction to this path causes a measurable performance degradation. Using the PROGRAM parameter adds at least 9 instructions to the enter path. (The enter path is even longer if a HASH chain needs to be processed.) If all ENTxCs were coded with the PROGRAM parameter, the performance could be reduced 5–8%. Monitor its use.
- A program that issues the ENTDC macro causes all programs assigned to the ECB to be released. All program levels are made available and the first program level is used for the specified program.

## ENTDC

- An ENTDC macro must not be issued between the executions of an ENTRC macro and a BACKC macro. If this sequence occurs, control is transferred to the system error routine.
- The specified program must have been allocated by the system allocator (see *TPF System Installation Support Reference*) or have been loaded online before the Enter was run. If not, a system error results.
- The specified program receives control in its allocated addressing mode. The operational program registers R0-R7 have the same value they had when the ENTDC macro was issued. The condition code and the contents of the scratch registers R14 and R15 are unpredictable.
- Users of the ALASC macro should note the programming considerations in the ALASC specifications relative to the ENTDC macro.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.

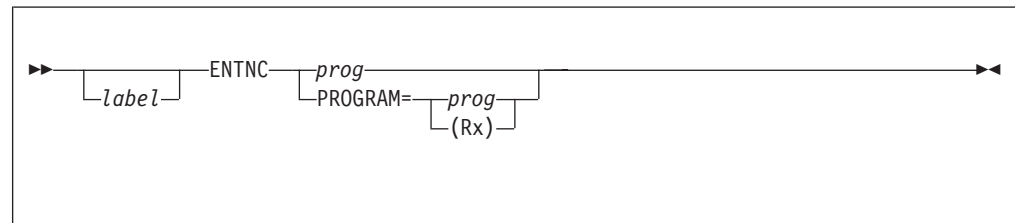
## Examples

None.

## ENTNC—Enter a Program with No Return Expected

This general macro transfers control to the specified operational program. The entering program stops.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be entered. This method generates a VCON resolved at link edit time into a Program Attribute Table (PAT) displacement. This is the preferred method for specifying the program name.

#### PROGRAM

The name of the program can alternately be provided via the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the PAT displacement. This method has a longer path length than the one described above.

*prog*

The name of the program that is to be entered.

*(Rx)*

A register (R0–R7) that contains the address of the program name.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

Control is never returned to a program which issues the ENTNC macro.

### Programming Considerations

- This macro can be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB being processed before using this macro.
- This macro may be used by control program OPZERO type code when activating ECB programs. The CP segment entering the ENTNC must be running in the SVM and have the SVA of the ECB in R9.
- Using the PROGRAM parameter sacrifices some performance. Its use should be monitored.
- This macro is used when a return is not expected to the current program. The ENTRC macro should be used when a return is anticipated.
- A program which issues the ENTNC macro is released from the ECB. The program level that is made available is used for the specified program.

## ENTNC

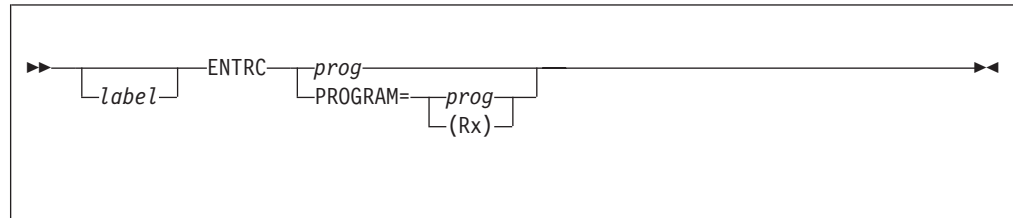
- The specified program must have been allocated by the system allocator (see to *TPF System Installation Support Reference*).
- The specified program receives control in its allocated addressing mode. The operational program registers R0–R7 have the same value they had when the ENTNC macro was issued. The condition code and the contents of the scratch registers R14 and R15 are unpredictable.
- Users of the ALASC macro should note the programming considerations in the ALASC specifications relative to the ENTNC macro.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.
- ENTNC cannot be called from an ISO-C segment (coded with BEGIN TPFISOC=YES).

## Examples

None.

This general macro transfers control to the specified operational program. The current active program remains held by the entry control block (ECB). The address of the next sequential instruction (NSI) in the current program is saved for an expected return.

## Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be entered. This method generates a V-con resolved at link edit time into a Program Attribute Table (PAT) displacement. This is the preferred method for specifying the program name.

## PROGRAM

The name of the program can alternately be provided using the PROGRAM parameter. This method generates constant data which will be used at execution time to determine the PAT displacement. This method has a longer path length than the one described above.

*prog*

The name of the program that is to be entered.

 $(Rx)$ 

A register (R0-R7) that contains the address of the program name.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The number of program nesting levels allowed is determined by a parameter in keypoint record A, symbolic tag CK1PLV. Accordingly, this macro cannot be used if the number of nesting levels is already at the maximum.

## Return Conditions

- Control is normally returned to the next sequential instruction (NSI). Control returns to the requesting program in the addressing mode in which it was operating prior to issuing the ENTRC macro.
- The contents of the operational program registers R0-R7 can be changed. These registers have the same values they had when the BACKC macro was issued to return control. Thus registers R0-R7 reflect any modifications made by the specified program (or by other programs it subsequently invokes).
- The contents of the scratch registers R14 and R15 are unpredictable.
- The condition code is unpredictable.

## ENTRC

### Programming Considerations

- The ECB reference register (R9) must contain the address of the ECB being processed before using this macro.
- This macro is used when a return is expected to the current program. The ENTNC macro should be used where no return is anticipated.
- ENTRC to FACE or FACS is a special type of enter and cannot be specified with the PROGRAM parameter. Attempting to use the PROGRAM parameter will result in a SNAP dump.
- Using the PROGRAM parameter sacrifices some performance. Its use should be monitored.
- The specified program must have been allocated by the system allocator (refer to *TPF System Installation Support Reference*).
- The specified program receives control in its allocated addressing mode. The operational program registers R0-R7 have the same value they had when the ENTRC macro was issued. The condition code and the contents of the scratch registers R14 and R15 are unpredictable.
- Users of the ALASC macro should note the programming considerations in the ALASC specifications relative to the ENTRC macro.
- In addition to the normal macro trace information the macro trace for this macro contains the name of the macro being returned to.

### Examples

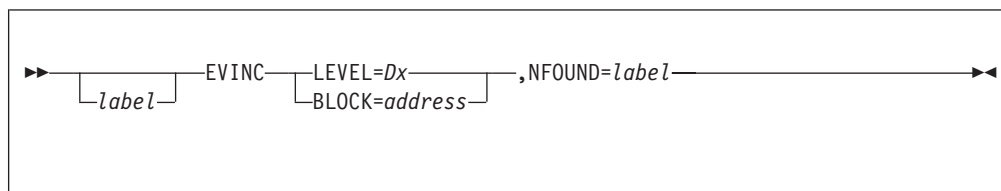
None.



## EVINC—Increment Count for Event

This general macro is used to increment the count dynamically for a count-type event after the event has been defined via the EVNTC macro. (See “EVNTC—Define Internal Event” on page 181.) An EVNWC macro cannot have been issued for the event.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**LEVEL=Dx**

A core block reference word and file address reference word (ranging from D0 to DF). The file address reference word on the specified data level must contain the 8-character name of the event.

**BLOCK=address**

The address of an area that contains the event block formatted as defined by EV0BK. The name of the event must reside in EVNBKN. The symbol you specify can be either an assembler label or a register enclosed in parentheses.

**Note:** LEVEL and BLOCK are mutually exclusive. One or the other is required.

**NFOUND=label**

A label where processing resumes if the event does not exist. This parameter is required.

### Entry Requirements

- When EVINC is coded with the LEVEL parameter, the symbolic name of the event to be waited for must be in CE1FAx (where x is the specified data level).
- When EVINC is coded with the BLOCK parameter, the symbolic name of the event must be in the field EVNBKN in the area pointed to by the BLOCK parameter.
- R9 must contain the address of the ECB being processed.

### Return Conditions

- Control will be returned to the next sequential instruction (NSI) with the count for the event incremented by one. If the event has not been defined via the EVNTC macro, the NFOUND branch will be taken.
- The contents of R14 and R15 are unknown. The contents of R0 — R7 are preserved across this macro call.

### Programming Considerations

- This macro may be executed on any I-stream.
- The specified event must be a count-type event. An EVNWC macro cannot have been issued for the event. If either of these conditions is violated, a system error is taken and the ECB exits.

## EVINC

- The maximum number of times the count for an event may be incremented is 32767.
- Defined events in MDBF systems are subsystem unique. The NFOUND branch will be taken if a match on the event name is found but the data base identifier (DBI) of the issuer is not the same as that of the subsystem that defined the event.

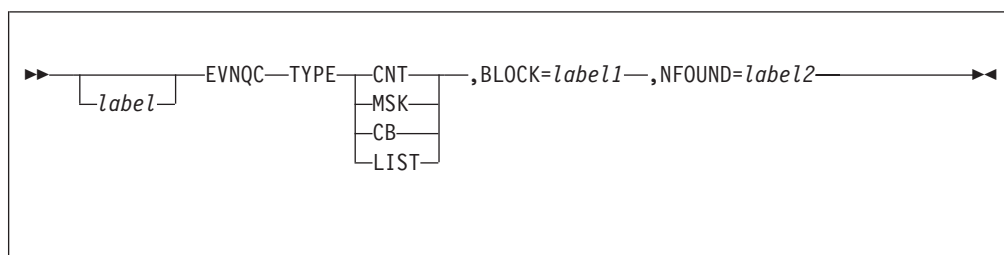
## Examples

None.

## EVNQC—Interrogate Event Status

This general macro interrogates the status of a specific event. It is used with macros for defining internal events (see “EVNTC—Define Internal Event” on page 181), waiting for events to complete (see “EVNWC—Wait for Event Completion” on page 187), and marking event completions (see “POSTC—Mark Event Completion” on page 306).

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### TYPE

The type of event being interrogated:

##### CNT

Count type event

##### MSK

Mask type event

##### CB

Core block oriented event.

##### LIST

List-type event.

#### BLOCK=*label1*

An area formatted as defined in EV0BK that contains the name of the event being queried. The area is used to return the current values of the specified event.

#### NFOUND=*label2*

A label where processing resumes if the event does not exist.

### Entry Requirements

- The symbolic name of the event being interrogated must be in the field EVNBKN in the area pointed to by the BLOCK parameter.
- R9 must contain the address of the ECB being processed.

### Return Conditions

- For the MSK- or CNT-type event, the area specified by the block parameter is initialized with the current mask or count value (EVNBKM1/EVNBKC1) and the contents of the accumulated POST MASK 2 field (EVNBKM2) from the event. (See “POSTC—Mark Event Completion” on page 306.)
- For the LIST-type event, the area specified by the block parameter is initialized with the contents of the current event list data, the number of data items (EVNBKLC), the size of a data item (EVNBKLS), and the list of data items

## EVNQC

(EVNBLKI). In addition, each list data item contains a flag field (EVNBKLIF), which indicates if the item has been posted, and an error indicator (EVNBKLIE) as coded in the POSTC macro.

- The EVNBKE field is set with the error indicator if an error occurs.

For an MSK-, CNT-, or CB-type event, the error indicator in the EVNBKE field is the value returned in the ERCODE parameter as coded in the POSTC macro. If the event is a CB type, the core block reference word (CBRW) information will not be returned. That information can only be obtained by issuing an EVNWC macro.

For a LIST-type event, the error indicator in the EVNBKE field is set to a generic error code value of X'7F'. The error indicator field (EVNBKLIE) for each list item contains the value returned in the ERCODE parameter as coded in the POSTC macro.

- Condition codes:
  - If the event has not completed, condition code 0 will be set in the PSW.
  - If the specified event has completed, condition code 1 will be set in the PSW.
  - Condition code 2 is reserved.
  - If the specified name is not found, condition code 3 is set in the PSW.

If an NFOUND parameter was coded and the name is not found, an immediate return is made to the specified label. Otherwise control is returned to NSI.
- The contents of R14 and R15 are unknown. The contents of R0-R7 are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- This macro causes no loss of control.
- The EVNQC macro will return any current information from an outstanding event but does not affect the state of the interrogated event. If the event has completed, an EVNWC macro is still required to cause the event to be deleted from the system.
- Any ECB is allowed to interrogate any existing event for its subsystem.
- For a CB type, only the EVNBKM1, EVNBKM2 and EVNBKE fields will be returned. The core block information will not be returned on an EVNQC macro. That information is only returned on a EVNWC macro.
- For a LIST type, the returned list is the same as the current event list. To determine if any of the items have been posted, it is necessary to interrogate each data item. If any of the items have been posted with an error, the gross error is set in field EVNBKE.

## Examples

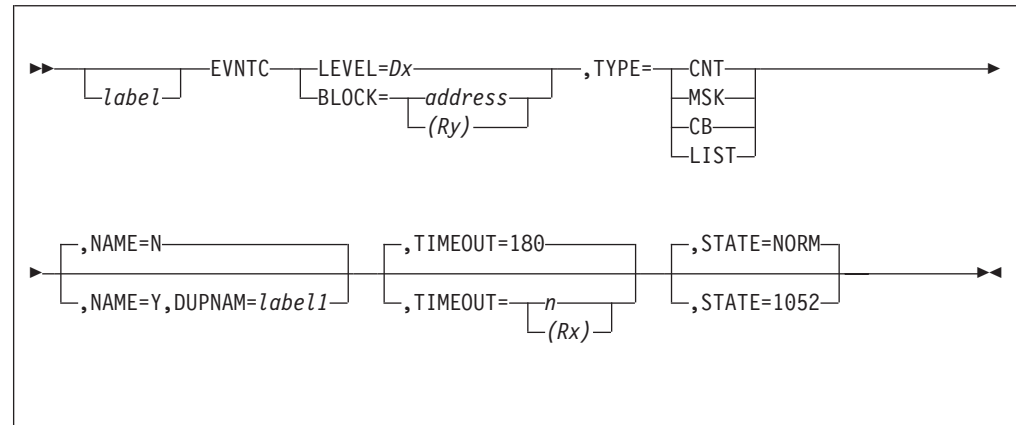
None.

## EVNTC—Define Internal Event

This general macro is used to define for the control program a named event that can be waited on by this ECB and posted by other ECBs. This macro is used with the POSTC (see “POSTC—Mark Event Completion” on page 306) and the EVNWC macros (see “EVNWC—Wait for Event Completion” on page 187).

You can use the EVNTC, POSTC, EVNWC, and SAWNC macros to pass the contents of a core block from one ECB to another ECB.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **LEVEL=Dx**

A core block reference word and file address reference word (D0-DF). LEVEL and BLOCK are mutually exclusive.

#### **BLOCK=address|(Ry)**

If specified, it is the address of an area which contains the EVNTC parameters. The area should be formatted as defined by the data area EV0BK. BLOCK and LEVEL are mutually exclusive. If a register is specified, it must be R0–R7 or R14–R15.

#### **TYPE**

The type of event being defined. Refer to Entry Requirements for more information about the TYPE parameter.

##### **CNT**

Event is a counter type. Each POSTC issued will cause the specified count to be decremented by one. When the count becomes 0, the event will be marked complete.

##### **MSK**

Event is mask type. Each POSTC issued will contain a 16 bit mask which will be used to reset the EVNTC specified mask bits. When the event's mask is completely reset, the event will be marked complete.

##### **CB**

Event is core block oriented, implies count of 1. The first POSTC will cause the event to be marked complete.

##### **LIST**

Event is oriented around a list of specified values. Each POSTC macro that

## EVNTC

is issued will contain a value that will be used to post the corresponding value in the EVNTC macro specified list. When all values are posted in the event list, the event will be marked completed.

**Note:** For TYPE=LIST, the BLOCK parameter must be specified.

### NAME=Y|N

If specified as Y, the event name is supplied by the issuer in the file reference word for this level or in the area specified by the BLOCK parameter. If not specified, or given as 'N', a unique name is generated by the macro processor and returned in the file reference word for the level or in the area specified by the BLOCK parameter.

### DUPNAM=*label*/1

This must be specified if NAME=Y is given. This is the label to branch to if the specified name already exists in the event table.

### TIMEOUT=*n*

If specified, the value in seconds an ECB can wait on the event before the event is assumed to be in error. The value may be specified either by a self-defining value (from 0 to 4096) or by a value in a register (from 0 to 32 768). If a value of 0 is specified, time-out is not performed. The default time-out is 180 seconds. If a register is specified, it must be R0–R7 or R14–R15.

### STATE

Indicates valid states for time-out processing. The default value is NORM.

#### NORM

Event will have time-out processing only when in NORM state. During time out processing the count specified by TIMEOUT will be decremented by one provided that the system state is NORM. If the value of TIMEOUT becomes 0, the event is posted as having timed out, regardless of system state. NORM is the default.

#### 1052

Event will timeout processing in all states with S-TIMERS on. During time out processing the count specified by TIMEOUT will be decremented by one. If the value of TIMEOUT becomes 0, then the event is posted as having timed out, regardless of system state.

## Entry Requirements

- EVNTC using LEVEL
  - The symbolic name to be assigned to the event, if supplied by the issuer, must be 8 characters in length, and must be in CE1FAx, where x is the specified level.
  - For CNT type, CE1CRx, bytes 0–1 must contain the count value for the event. This is a 16-bit number.
  - For MSK, CE1CRx, bytes 0–1 must contain the 16-bit mask value to be used for the event.
  - For CB, CE1CRx is not used.
- EVNTC using BLOCK
  - The symbolic name to be assigned to the event, if supplied by the user, must be 8 characters long, and must be in EVNBKN.
  - For CNT type, EVNBKC1 must contain the count value for the event. This is a 16-bit number.

- For MSK EVNBKM1 must contain the 16-bit mask value to be used for the event.
- For CB, there is no area to be set up.
- For LIST, EVNBKLC must contain the number of items in the data list, EVNBKLS must contain the size of a data item, and EVNBKLI must contain the list of data items.

**Note:** Using the GENLC macro to create or modify the list will ensure that these fields are defined correctly. See “GENLC–Generate a Data List” on page 237 for more information.

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

## Return Conditions

- Control is returned to the next sequential instruction unless the specified name is not unique, in which case the label for DUPNAM is branched to.
- If the LEVEL parameter was used, the value of the core reference word is unchanged and the file reference word will contain the assigned name for the event. The level is available for reuse.
- If the BLOCK parameter was used, the EV0BK area is unchanged unless NAME=N was specified, in which case the field EVNBKN will contain the assigned name for the event.
- The contents of R14 and R15 are unknown. The contents of R0-R7 are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- Defined events in MDBF systems are subsystem unique.
- To ensure that an event ends, use an EVNWC or SAWNC macro.
- If a count-type event is dynamically incremented using the EVINC macro, use the EVNWC or SAWNC macro after the final count is defined. (See “EVINC–Increment Count for Event” on page 177.)
- The level that is used must not be holding a block. If it is, a system error occurs and the ECB exits.
- You can use the EVNTC macro and either the EVNWC or SAWNC macro to delay an ECB for a specified number of seconds. When the EVNWC or SAWNC macro is processed, the ECB will be delayed by the amount of time specified by the TIMEOUT parameter of the EVNTC macro. You do not have to code the POSTC macro.
- Use the GENLC macro to generate the event data list (see “GENLC–Generate a Data List” on page 237 for more information).
- The EV0BK DSECT describes the user area that is used as a parameter to the following macros:

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <b>EVNTC</b> | See “EVNTC–Define Internal Event” on page 181.                   |
| <b>EVNWC</b> | See “EVNWC–Wait for Event Completion” on page 187.               |
| <b>POSTC</b> | See “POSTC–Mark Event Completion” on page 306.                   |
| <b>SAWNC</b> | See “SAWNC–Wait for Event Completion, Signal Aware” on page 345. |
| <b>GENLC</b> | See “GENLC–Generate a Data List” on page 237.                    |

## EVNTC

The use of a user area defined by the EV0BK DSECT allows the application to use the above macros without needing a free data level. If this user area is not used, the application must have an available data level. The user area defined by the EV0BK DSECT is used to pass to the SVC handler for these macros the information needed to perform the specified function.

The name field is required for the EVNTC, POSTC, EVNWC, and SAWNC macros. Other fields in the EV0BK DSECT may be required according to the macro that is being issued.

- The event area is allocated and created by the user application program. It consists of an area of storage and exists only as long as that area of storage is owned by the issuing application program.
- The basic EV0BK block requires 16 bytes of storage. The extended EV0BK block, one for which a data list is defined, can require up to a maximum of 424 bytes. EV0BK must start on a fullword boundary.
- The content of the user event area is as follows:

| Field    | Description                                                                                                                                                                                                                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EVNBKN   | This is the name of the event. It is can either be supplied by the application or dynamically created by the EVNTC macro. For POSTC and EVNWC, the name must be supplied.                                                                                                                                                                           |
| EVNBKC1  | This must be set by the application for a count type of event. This value specifies the number of POSTC macros that must be issued before the event is considered satisfied.                                                                                                                                                                        |
| EVNBKM1  | This field is used both when creating a mask type of event and also when posting the same event. When creating the event, this field supplies the master mask, which must be reset to 0 before the event is to be considered satisfied. For a POSTC macro, EVNBKM1 specifies the bits to be reset in the mask specified when the event was created. |
| EVNBKM2  | This is a 16-bit mask which can be used for informational purposes and is OR'd into an accumulator and returned to the application when the event has been satisfied. It is referred to as the POST MASK 2 field in the POSTC and EVNWC macro descriptions.                                                                                         |
| EVNBKLC  | This field is used when creating a list-type event and also when posting the same event. This field will contain the number of items contained in the data list.                                                                                                                                                                                    |
| EVNBKLS  | This field is used when creating a list-type event and also when posting the same event. This field will contain the size, in bytes, of a data list item.                                                                                                                                                                                           |
| EVNBKLI  | This field is used when creating a list-type event and also when posting the same event. This field will contain the list of data items. Each item in the list has the following format:                                                                                                                                                            |
| EVNBKLIF | This is the flag byte; it indicates the status of the data item.                                                                                                                                                                                                                                                                                    |
| EVNBKLIE | This is the error value if the item is posted with an error.                                                                                                                                                                                                                                                                                        |
| EVNBKVLS | This is an optional user-defined return value area.                                                                                                                                                                                                                                                                                                 |



EVNBKLID This is the list item data.

EVNBKE This field is used by the EVNWC macro for returning the error value when an event has been posted with an error.

For CNT, MSK, and CB event types, the value in this field is set by the application when it issues the POSTC macro. The value can be any combination of bits except X'80'. The X'80' value is reserved by the TPF 4.1 system to present a timeout error indication.

For a LIST event type, issuing a POSTC macro indicating an error for any of the list items causes the EVNBKE field to be set to X'7F'. The error value for the list item in error is set in the EVNBKLIE field for that list item.

EVNBKL Is the size of the area defined by the EV0BK DSECT, which is currently 16 bytes.

The following table indicates the fields used and or modified by the event macros:

| Field Name | EVNTC             | POSTC             | EVNWC    | SAWNC    |
|------------|-------------------|-------------------|----------|----------|
| EVNBKN     | Optional          | Required          | Required | Required |
| EVNBKC1    | Possibly Required |                   |          |          |
| EVNBKM1    | Possibly Required | Possibly Required | Modified | Modified |
| EVNBKM2    |                   | Optional          | Modified | Modified |
| EVNBKLC    | Possibly Required | Possibly Required | Modified | Modified |
| EVNBKLS    | Possibly Required | Possibly Required | Modified | Modified |
| EVNBKLI    | Possibly Required | Possibly Required | Modified | Modified |
| EVNBKE     |                   |                   | Modified | Modified |

**Notes:**

1. Field names are defined in the EV0BK DSECT.
2. Fields marked "Possibly Required" depend on the type of event being specified.
3. Fields marked "Modified" are modified by the SVC Handler.
4. Fields with no indication are ignored.

## Examples

- A count-type event is defined for an ECB using the count value in the first 2 bytes of CE1CR5.  
EVNTC LEVEL=D5,TYPE=CNT
- The same count-type event is defined but the time-out value is specified as 10 seconds using the TIMEOUT parameter.  
EVNTC LEVEL=D5,TYPE=CNT,TIMEOUT=10
- The address for an EV0BK formatted block is specified in R1 for a count-type event. The time-out in seconds is specified in R7. The name for the event is

## EVNTC

specified in the EVNBKN field of EV0BK. If the specified name is already associated with an event, control transfers to label ERROR1.

```
EVNTC BLOCK=(R1),TYPE=CNT,NAME=Y,DUPNAM=ERROR1,TIMEOUT=(R7)
```

- The address of an EV0BK formatted block is specified in R15 for a mask-type event. EV0BK field EVNBKM1 contains the mask. A time-out value is specified in R14.

```
EVNTC BLOCK=(R15),TYPE=MSK,TIMEOUT=(R14)
```

- A list of active processors is generated for creating an event. The list-type event is defined by using the EV0BK block specified in R4.

```
GENLC CREATE,BLOCK=(R4),DATA=PROCESSOR,INCLUDE=ACTIVE
```

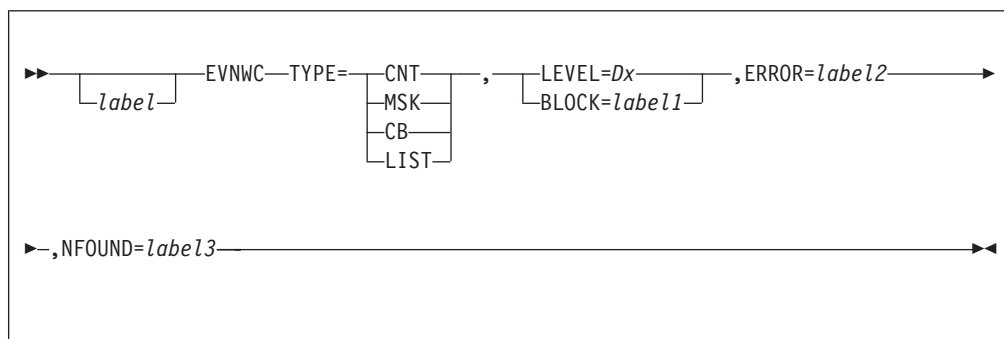
```
EVNTC BLOCK=(R4),TYPE=LIST
```

## EVNWC—Wait for Event Completion

This general macro is used to wait for the completion of a named event. It is used with the EVNTC (see “EVNTC—Define Internal Event” on page 181) and POSTC (see “POSTC—Mark Event Completion” on page 306) macros.

The EVNTC, POSTC and EVNWC macros can be used to pass the contents of a core block from one ECB to another ECB.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### TYPE

The type of event being waited for:

##### CNT

Count type event.

##### MSK

Mask type event.

##### CB

Core block oriented event.

##### LIST

List-type event.

**Note:** See the POSTC macro for more information about these parameters.

For TYPE=CB, LEVEL must be specified and BLOCK may be specified. If BLOCK is given, the event name is retrieved from the area specified and the error code if one, is returned in the specified area.

For TYPE=LIST, the BLOCK parameter must be specified.

#### BLOCK=label1

An area formatted as defined in EV0BK that contains the name of the event being waited on. The area is used to return the values from the completed event. LEVEL and BLOCK are mutually exclusive except for TYPE=CB.

#### LEVEL=Dx

A core block reference word (CBRW) and file address reference word (D0-DF). LEVEL and BLOCK are mutually exclusive except for TYPE=CB.

#### ERROR=label2

A label to branch to if the event has completed with error.

## EVNWC

**NFOUND=***label***3**

A label to branch to if the event does not exist.

## Entry Requirements

- For an EVNWC macro using the BLOCK parameter:
  - The symbolic name of the event you are waiting for must be in field EVNBKN in the area pointed to by the BLOCK parameter. For TYPE=LIST, the full size of EV0BK must be allocated to ensure that there is enough room for the returned data list. For more information about the EV0BK area, see “EVNTC—Define Internal Event” on page 181.
- For an EVNWC macro using LEVEL without the BLOCK parameter
  - The symbolic name of the event to be waited for must be in CE1FAx, where x is the specified level.
- R9 must contain the address of the ECB being processed.

## Return Conditions

- For an EVNWC macro with the BLOCK parameter:
  - The area specified will be filled as it relates to the specified event type. (See “POSTC—Mark Event Completion” on page 306.)
    - If it is an MSK type, the area specified is filled in with the remaining mask and the contents of the accumulated POST MASK 2 field (EVNBKM2) from the event.
    - If it is a CNT type, the area specified is filled in with the remaining count value.
    - If it is a CB type, the CBRW specified by the LEVEL parameter is set to the value passed by the POSTC macro. Enter the LEVTA macro to determine if a core block was passed.
    - If it is a LIST type, the area specified is filled in with the data list information. To determine the status of the data list items, the flag field (EVNBKLIF) of each item must be interrogated.
  - The EVNBKE field is set with the error indicator if an error occurs.
    - For the MSK-, CNT-, or CB-type event, the error indicator in the EVNBKE field is the value returned in the ERCODE parameter, as coded in the POSTC macro.
    - For the LIST-type event, the error indicator in the EVNBKE field is set to a generic error code value of X'7F'. The error indicator field (EVNBKLIE) for each list data item contains the value returned in the ERCODE parameter as coded in the POSTC macro.
- For an EVNWC macro with the LEVEL parameter only:
  - The core block reference word of the specified level is set to the remaining mask or count value, CE1CRx bytes 0-1, and to the contents of the accumulated POST MASK 2 field, CE1CRx bytes 3-4. (See “POSTC—Mark Event Completion” on page 306). The CE1SUD field for the level will be set with the error indicator if an error occurred. The error indicator in the CE1SUD field is the returned ERCODE parameter as coded in the POSTC macro. If it is a ‘CB’ type, the full core block reference word is set to the value passed by the POSTC macro. A LEVTA macro should be issued to determine if a core block was passed.
- If the specified name is not found an immediate return is made to the label specified by the NFOUND parameter. If the event completed with error, control is returned to the label specified by the ERROR parameter. Otherwise control is returned to NSI.

- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- The EVNWC macro arms the named event to allow event completion to be posted to the ECB. The EVNWC macro increments the I/O counter and flags the named event so that on completion, the event can be posted to the ECB. If the named event has completed, the EVNWC macro initiates the posting process.
- Defined events in MDBF systems are subsystem unique.
- Issuance of the EVNWC macro causes an unconditional loss of control for the issuing ECB unless the named event is not found.
- More than one ECB is allowed to wait for the same named event except for type CB. When the event completes all waiting ECBs are posted complete. For type 'CB', only the creating ECB may wait on the event. Any other ECB attempting to wait on the event will be returned with a not-found condition.
- The EVNTC and EVNWC macros can be coded to delay an ECB for a given amount of seconds. This essentially will **delay** the ECB until the amount of time specified in EVNTC TIMEOUT=xx has elapsed. The POSTC macro does not need to be coded.

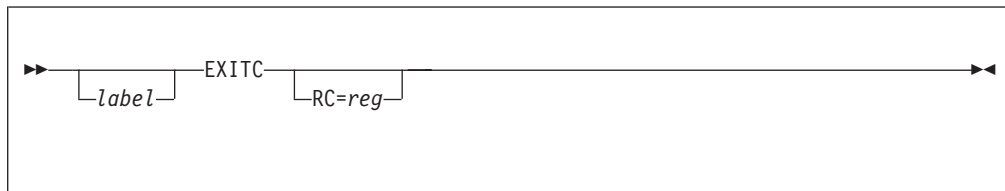
## Examples

None.

## EXITC—Processing of an Entry Is Complete

This general macro causes the control program to release the entry control block (ECB), any working storage, or program record blocks held by the ECB, thereby ending the life of the entry in the system.

### Format



*label*

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

**RC=reg**

specifies the general register (R0–R7, R14, or R15) that contains the return code to be passed back to the parent ECB.

### Entry Requirements

- All held records must be released before executing an EXITC.
- The ECB reference register (R9) contains the address of the ECB being processed before using this macro.

### Return Conditions

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The ECB and all attached core storage blocks will be released.
- No return is made to the operational program.

### Programming Considerations

- This macro can be executed on any I-stream.
- Following the use of this macro, no return is made to the operational program. This operational program may be executed for processing of other messages. Accordingly, it may be necessary to reset temporary counters, program switches, or other ECB-controlled resources to allow for proper execution of this program.
- If the ECB using the EXITC macro is currently holding any file record, control is transferred to the system error routine.
- If the ECB using the EXITC macro currently has a commit scope open, control is transferred to the system error routine and processing ends as if a TXRBC macro was entered.

### Examples

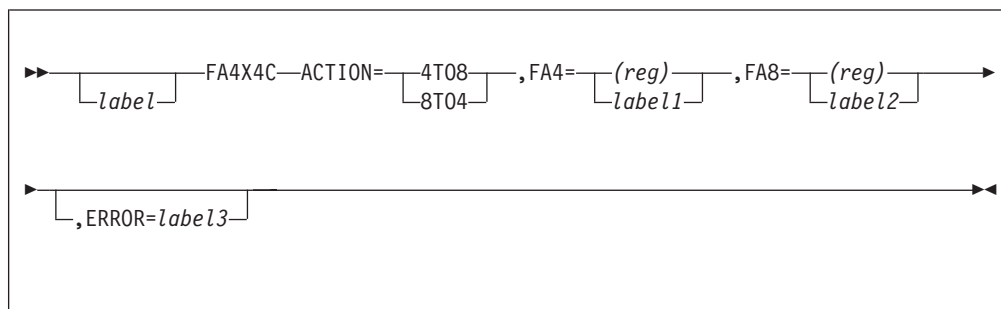
None.

## FA4X4C—Convert a File Address

Use this general macro to do the following:

- Convert a 4-byte file address to an 8-byte file address in 4x4 format.
- Convert an 8-byte file address in 4x4 format to a 4-byte file address.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**FA4=(reg)|label1**

The label or general register (R1–R7, R14, R15) containing the location of a 4-byte file address.

**FA8=(reg)|label2**

The label or general register (R1–R7, R14, R15) containing the location of an 8-byte file address.

#### **ACTION**

Specifies the type of file address conversion to perform.

##### **4TO8**

Indicates that the FA4 parameter contains a 4-byte file address, which is used as input to the file address conversion. The corresponding 8-byte file address in 4x4 format will be returned in the location represented by the FA8 parameter.

##### **8TO4**

Indicates that the FA8 parameter contains an 8-byte file address in 4x4 format, which is used as input to the file address conversion. The corresponding 4-byte file address will be returned in the location represented by the FA4 parameter.

**ERROR=label3**

The label in which to branch if an error occurs during macro processing.

### Entry Requirements

None.

### Return Conditions

- All registers are maintained across this macro call.
- Control is returned to the next sequential instruction (NSI) or to the label specified on the ERROR parameter if an error occurs.

## **FA4X4C**

### **Programming Considerations**

- This macro can be run on any I-stream.
- If the file address specified for the FA8 parameter is not in 4x4 format and ACTION=8TO4 is specified, control will be passed to the system error routine and the ECB will be exited unless the ERROR parameter is coded.

### **Examples**

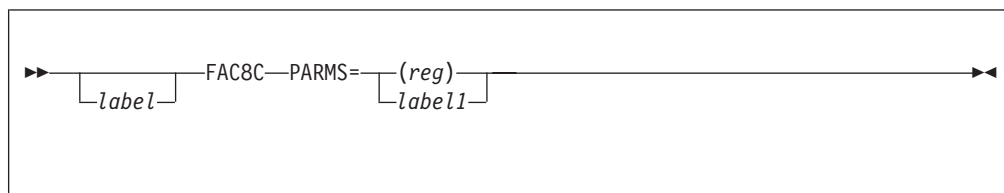
None.



## FAC8C—Calculate an 8-Byte File Address

Use this general macro to provide the interface to file address compute program (FACE) address generation routines for programs that use a data event control block (DECB) or want an 8-byte file address. The service is similar to calling the file address compute program (FACE or FACS) segment.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**PARMS=(reg)|label1**

The label or general register (R1–R7) containing the address of a parameter block as described by the IFAC8 DSECT. Fields in the input area of this block that are to be provided by the caller must be initialized before the macro is called.

### Entry Requirements

- PARMS=(reg) must specify a register containing the address of a block of contiguous, addressable storage as described by the IFAC8 DSECT.
- R9 must contain the address of the entry control block (ECB) being processed.
- Field IFACTYP in the parameter block specified by the PARMS parameter must be set to indicate if the record type specified is symbolic (IFACFCS) or numeric (IFACFCE).

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- Registers R0–R7 will be maintained. The contents of R14 and R15 cannot be predicted.
- Output area fields will be filled in by the macro service routine as specified in the IFAC8 DSECT.
- Inspect the return code field (IFACRET) to determine the results of the call.
- The file address will be returned if the return code in the parameter block indicates a normal return.

Return conditions are found through the IFAC8 DSECT in the IFACRET field. The following conditions are represented in the IFACRET field:

| Condition Name | Condition Value | Description of Error                                       |
|----------------|-----------------|------------------------------------------------------------|
| IFACNRM        | 0               | Normal return                                              |
| IFACNIU        | 1               | Record type is not in use                                  |
| IFACRTH        | 2               | Record type does not exist or exceeds the FACE table limit |
| IFACROR        | 3               | Record ordinal number is outside the allowable range       |

## FAC8C

|         |   |                                                |
|---------|---|------------------------------------------------|
| IFACNSP | 4 | No split chain for the record                  |
| IFACPOR | 5 | Input parameter is outside the allowable range |

- The IFACMAX field contains the largest ordinal number for the referenced record type. If IFACMAX is zero, an error occurred that is reported in IFACRET.
- The file address reference format (FARF) address returned is for the subsystem and subsystem user (SSU), processor, and I-stream that the entry control block (ECB) is running in 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 in IFACMAX (unless the requested ordinal is higher than the record type allows; zero is then returned).

## Programming Considerations

- This macro can be run on any I-stream.
- This macro can only be used by E-type programs.

## Examples

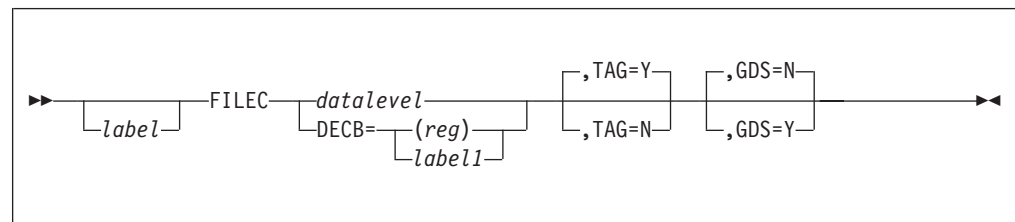
None.

## FILEC—File a Record

Use this general macro to write a record to a file from storage. The block of storage referenced at the specified entry control block (ECB) data level or data event control block (DECB) is removed from the ECB. The control program writes the record on the duplicate copy if one exists. This macro can be used to file a record that resides in either VFA or on an external device.

The FILEC macro returns the block of storage to the appropriate pool that is referenced in the core block reference word (CBRW) at the specified ECB data level or DECB.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

### **DECB=(reg)||label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

### **TAG**

Specify one of the following:

- Y** The name of the segment that issued the FILEC is put into the storage block to be filed.
- N** This field is not set.

The default is Y.

### **GDS**

Specify one of the following:

- N** The file address in the specified ECB data level or DECB is for a record from the online database.
- Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A block of storage must be held by the ECB at the specified ECB data level or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the file address reference format (FARW) for the specified ECB data level or DECB.

## FILEC

- The GDSRC macro must be executed to set up the file address before FILEC is used to access a general data set record.

## 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.
- 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 specified CBRW is initialized to indicate that a block of storage is no longer held.
- The FARW at the specified ECB data level (*datalevel*) or DECB is unchanged.
- The program identification was placed in the header of the record unless TAG=N was specified.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is holding a block of storage at the specified ECB data level or DECB and the file address contained at the specified ECB data level or DECB is valid. In addition, the record ID at the specified ECB data level or DECB is checked with the record ID in the record. If any condition is violated, control is transferred to the system error routine. When the RCC in the FARW is zero, the control program ignores verification of the RCC. If the RCC is nonzero, the control program verifies that the code specified in the FARW is the same as the code in the header of the record. If the codes are not equal, control is transferred to the system error routine.
- The block of storage containing the data to be stored is no longer available to the operational program.
- The status of the operation can never be determined by the operational program.
- The operational program can use the specified CBRW immediately upon return from the control program.
- If FILEC is issued from a utility program, specifying TAG=N prevents the utility program name from being inserted in the storage block for filing. This allows the segment name performing the data manipulation to be inserted before the call to the utility routine is made, FILEC is issued, and the name is filed with the data.
- You cannot call the FILEC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record x
  3. TXSPC
  4. FILEC record x.
- TPF transaction services processing affects FILEC macro processing in the following ways:
  - The FILEC actions that are controlled by the record ID attribute table (RIAT) indicators are set at FILEC time (when the record is filed), but do not take place until the record has been committed.

- Hardening (that is, writing) to the DASD surface will occur only at commit time.
- FILEC macro changes are discarded and hardening does not occur after a TXRBC macro is called.
- If a system error occurs because of an ID or record code check, processing ends as if a TXRBC macro was called.
- Files to general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
- 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

None.

## FILKW–File Keyword

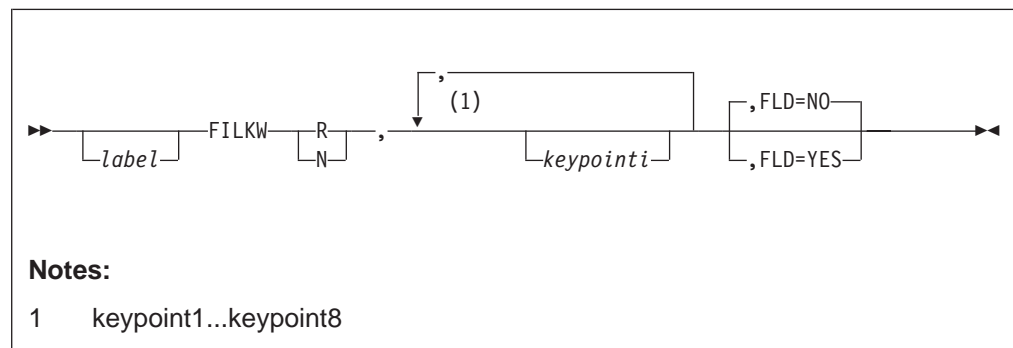
This general macro enables the filing of global keypointable fields or records.

This macro allows the application programmer to request the updating of keypoint records by specifying either the keypoint record name, or a field name in a keypoint record. It also provides the option of restoring the storage protection key in the PSW for the entry to match the application working storage key.

FILKW uses the request keypoint update macro (GLOUC), optionally preceded by the Restore Protection Key macro (KEYRC).

See also the global area program material in the *TPF System Installation Support Reference*.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

**R** Restore protection key.

**N** Do not restore protection key.

### *keypoint1,keypoint2,...keypoint8*

Names of those keypoint records to be filed,

- as they appear in the GLOBA macro, GLOBY macro or
- as they appear in keypoint records to be filed.

### **FLD**

Specify one of the following:

#### **YES**

One or more parameters keypoint1 through keypoint8 are field names.

#### **NO**

Only names of keypoint records appear in keypoint1 through keypoint8. This is the default.

## Entry Requirements

- The global fields must be defined, and have a valid base, by the use of the GLOBZ macro.
- If the field name facility (FLD=YES) is to be used, the field names must appear in a table in the FILKW macro definition and must be equated to the corresponding keypoint record name.

## Return Conditions

- Control is returned to the next sequential instruction.
- If the R parameter is used, the application program protection key will have been restored. The N parameter will not alter the protection key.
- If keypointing is active, those keypoints referenced as parameters (keypoint1 through keypoint8) will have been updated on file.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The condition code is unchanged.

## Programming Considerations

This macro can be executed on any I-stream.

## Examples

```
LABX FILKW R,@GBLCC,@VXIXV,FLD=YES
```

At label LABX the call to FILKW requests an update of keypoint @GBLCC at field @VXIXV. The protection key is restored to its normal value and @VXIXV is flagged by the FLD parameter as being a field (as opposed to a keypoint).

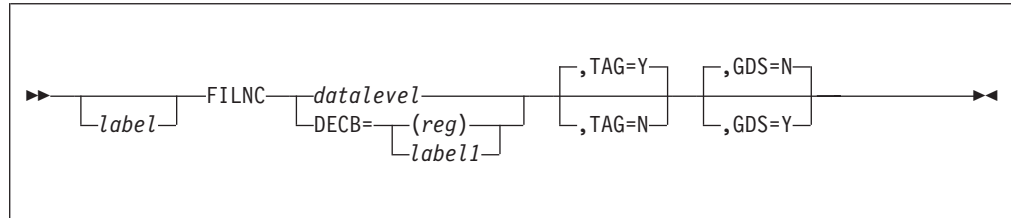
**Note:** Assume that @VXIXV is a field name in keypoint @GBLCC.

## FILNC—File a Record with No Release

This general macro writes a record to a file from storage. The block of storage referred to at the specified ECB data level or data event control block (DECB) is not removed from the entry control block (ECB). The control program writes the record on the duplicate copy if one exists. This macro can be used to file a record that resides in either the virtual file access (VFA) or on an external device.

The FILNC macro does not return a block of storage to any storage pool.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

### **DECB=(reg)||label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

### **TAG**

Specify one of the following:

- Y** The name of the segment that issued the FILNC is put into the storage block to be filed.
- N** This field is not set.

The default is Y.

### **GDS**

Specify one of the following:

- N** The file address in the specified ECB data level or DECB is for a record from the online database.
- Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A block of storage must be held by the ECB at the specified ECB data level (*datalevel*) or DECB.
- A file address, record ID, and a record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be executed to set up the file address before FILNC is used to access a general data set record.



## 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.
- 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 (*datalevel*) or DECB is unchanged.
- The status of the operation is unknown.
- The program identification was placed in the header of the record unless TAG=N was specified.
- The block of storage at the specified ECB data level (*datalevel*) or DECB is not available. The control program makes it available on return the next time the WAITC macro is run.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is holding a block of storage at the specified ECB data level or DECB. In addition, the record ID at the specified ECB data level or DECB is checked with the record ID in the record. If any condition is violated, control is transferred to the system error routine. If an incorrect file address is detected, the error indicator is set (CE1SUD or IDECSUD) and control is returned to the calling program. When the RCC in the FARW is zero, the control program ignores verification of the RCC. If the RCC is nonzero, the control program verifies that the code specified in the FARW is the same as the code in the header of the record. If the codes are not equal, control is transferred to the system error routine.
- The operational program must not use the specified ECB data level (*datalevel*) or DECB on return from the control program.
- To ensure completion of the operation a WAITC macro must be executed.
- An error is posted at WAITC time only if neither copy of the record was successfully updated.
- If the FILNC is issued from a utility program, specifying TAG=N prevents the utility program name from being inserted in the storage block for filing. This allows the segment name performing the data manipulation to be inserted before the call to the utility routine is made, FILNC is issued and the name is filed with the data.
- You cannot call the FILNC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILNC record x
  3. TXSPC
  4. FILNC record x.
- TPF transaction services processing affects FILNC macro processing in the following ways:
  - The FILNC actions that are controlled by the record ID attribute table (RIAT) indicators are set at FILNC time (when the record is filed), but do not take place until the record has been committed.

## FILNC

- Hardening (that is, writing) to the DASD surface will occur only at commit time.
- When a group of updated records is hardened to DASD, the filing sequence that was originally specified by the application program is maintained. Therefore, applications that rely on a specific sequence of FILNC and WAITC macros to control the order of updates in a chain of related records will not lose this ability when placed in a commit scope.
- FILNC macro changes are discarded and hardening does not occur after a TXRBC macro is called. The DASD surface remains unchanged.
- If a system error occurs because of an ID or record code check, processing ends as if a TXRBC macro was called.
- The WAITC macro does not report hardware-related filing errors when the FILNC macro is called in a commit scope. The block of storage at the specified level is available on return from the FILNC macro. Commit processing issues an implied WAITC macro.
- Files to general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
- 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

None.

## FILSC—File a Single Record

This general macro writes a record to a file from storage. An option provides the user with the facility to write either the primary or duplicate record only.

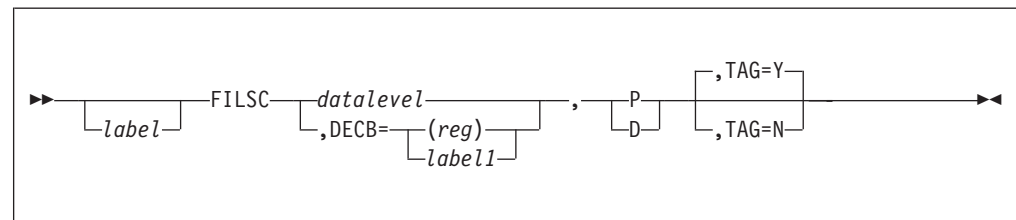
This macro is for filing records to the online database only and cannot be used to file records to either general files or general datasets.

The block of storage referred to at the specified entry control block (ECB) data level or data event control block (DECB) is removed from the ECB.

This macro causes 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.

The FILSC macro returns the block of storage to the appropriate pool that is referenced in the core block reference word (CBRW) at the specified ECB data level or DECB.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

### **DECB=(reg)||label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

**P** File primary record.

**D** File duplicate record.

### **TAG**

Specify one of the following:

**Y** The name of the segment that issued the FILSC macro is put into the storage block to be filed.

**N** This field is not set.

The default is Y.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A block of storage must be held by the ECB at the specified ECB data level (*datalevel*) or DECB.

## FILSC

- A file address, record ID, and record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.

## 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.
- 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 specified CBRW is initialized to indicate that a block of storage is no longer held.
- The status of the operation is unknown.
- The FARW at the specified ECB data level (*datalevel*) or DECB is unchanged.
- The program identification was placed in the header of the record unless TAG=N was specified.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is holding a block of storage at the specified ECB data level or DECB, the file address contained at the specified ECB data level or DECB is valid, and a duplicate exists when it is requested. In addition, the record ID at the specified ECB data level or DECB is checked with the record ID in the record. If any condition is violated, control is transferred to the system error routine. When the RCC at the specified ECB data level or DECB is zero, the control program ignores verification of the RCC. If the codes are not equal, control is transferred to the system error routine.
- The block of storage containing the data to be stored is no longer available to the operational program.
- The status of the output operation cannot be determined by the operational program.
- The operational program may use the specified CBRW immediately upon return from the control program.
- In the event of a noncorrectable error, the control program will not service the request from the alternate device. For example, if the request for the primary failed, the duplicate will not be updated.
- If the FILSC is issued from a utility program, specifying TAG=N prevents the utility program name from being inserted in the storage block for filing. This allows the segment name performing the data manipulation to be inserted before the call to the utility routine is made, FILSC is issued, and the name is filed with the data.
- 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

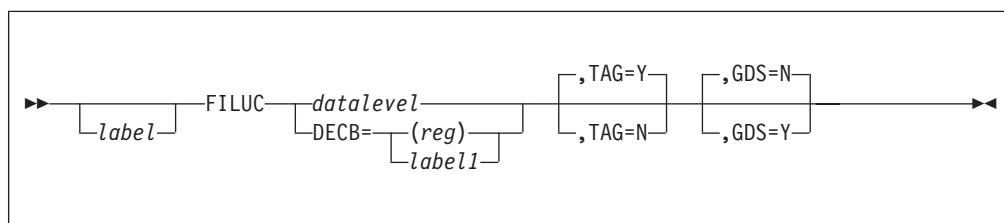
None.

## FILUC—File and Unhold a Record

This general macro writes a record to a file from storage and releases it from the exclusive control of the entry control block (ECB). The block of storage referred to at the specified ECB data level or data event control block (DECB) is removed from the ECB. The control program writes the record on the primary copy and the duplicate copy if one exists. This macro can be used to file a record that resides in either a virtual file access (VFA) or on an external device, depending on the system configuration. (See virtual file access information in *TPF Database Reference* for details of how FILUC processes a VFA record.)

The file address contained in the specified file address reference word (FARW) is released. Sometimes in TPF documentation the word unhold is used to mean released.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

#### **DECB=(reg)||label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

#### **TAG**

Specify one of the following:

**Y** The name of the segment that issued the FILUC is put into the storage block to be filed.

**N** This field is not set.

The default is Y.

#### **GDS**

Specify one of the following:

**N** The file address in the specified ECB data level or DECB is for a record from the online database. N is the default.

**Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A block of storage must be held by the ECB at the specified ECB data level (*datalevel*) or DECB.

## FILUC

- A file address that is being held now, the record ID, and the record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be executed to set up the file address before FILUC is used to access a general data set record.

## 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.
- 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 specified core block reference word (CBRW) is initialized to indicate that a block of storage is no longer held.
- The FARW at the specified ECB data level (*datalevel*) or DECB is unchanged.
- The file address contained in the specified FARW has been unheld. If another request was waiting to hold the same record, it has been serviced.
- The program identification was placed in the header of the record unless TAG=N was specified.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is holding a block of storage at the specified ECB data level or DECB and if the file address contained at the specified ECB data level or DECB is valid and is presently being held by this ECB. In addition, the record ID type at the specified ECB data level or DECB is checked with the record ID in the record. If any condition is violated, control is transferred to the system error routine. When the RCC at the specified ECB data level or DECB is zero, the control program ignores verification of the RCC. If the RCC is nonzero, the control program verifies that the code at the specified ECB data level or DECB is the same as the code in the header of the record. If the codes are not equal, control is transferred to the system error routine.
- The block of storage containing the data to be stored is no longer available to the operational program.
- The status of the operation can never be determined by the operational program.
- The operational program can use the specified CBRW immediately upon return from the control program.
- If FILUC is issued from a utility program, specifying TAG=N prevents the utility program name from being inserted in the storage block for filing. This allows the segment name performing the data manipulation to be inserted before the call to the utility routine is made, FILUC is issued, and the name is filed with the data.
- You cannot call the FILUC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record x
  3. TXSPC
  4. FILUC record x.

- TPF transaction services processing affects FILUC macro processing in the following ways:
  - The FILUC actions that are controlled by the record ID attribute table (RIAT) indicators are set at FILUC time (when the record is filed and unheld), but do not take place until the record has been committed.
  - Hardening (that is, writing) to the DASD surface will occur only at commit time.
  - FILUC macro changes are discarded and hardening does not occur after a TXRBC macro is issued. The DASD surface remains unchanged.
  - If a system error occurs because of one of the previous considerations, processing ends as if a TXRBC macro was called.
  - The file address appears to be unheld from the program point of view on return from the FILUC macro. For requests that come from outside of the commit scope, the file address still appears to be held. When the commit request is completed successfully, the file address is unheld and any waiting requests are serviced.
  - Files to general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
- 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

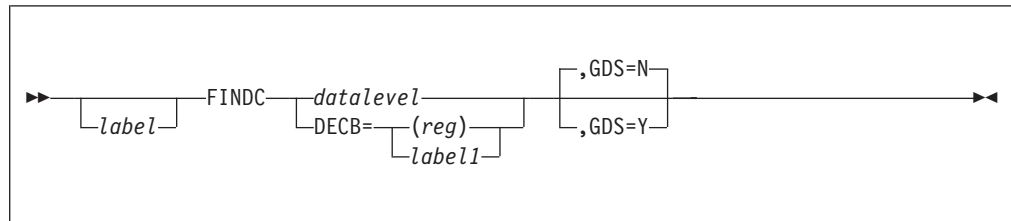
None.

## FINDC—Find a File Record

This general macro reads a record from a file into storage. A block of storage is obtained by the control program and reference to it is stored in the core block reference word (CBRW) at the specified time.

The requested record can be contained in the virtual file access (VFA).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*datalevel*

An entry control block (ECB) data level (D0–DF) that identifies the file address and core block information for the I/O request.

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB), which specifies the file address and core block information for the I/O request.

**GDS**

Specify one of the following:

- N** The file address in the specified ECB data level or DECB is for a record from the online data base. N is the default.
- Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

### Entry Requirements

- 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 (*datalevel*) or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the FARW for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be run to set up the file address before FINDC is used to access a general data set record.

### 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.
- 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 status of the operation is unknown.
- The contents of the CBRW at the specified ECB data level (*datalevel*) or DECB is unknown.
- The file address reference word (FARW) at the specified ECB data level (*datalevel*) or DECB is unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is not holding a block of storage at the specified ECB data level or DECB and if the file address contained at the specified ECB data level or DECB is valid. If either condition is violated, control is transferred to the system error routine. In addition, the control program verifies that the record ID at the specified ECB data level or DECB is equal to the record ID in the record. If the record ID specified is zero, this check is not made. The RCC at the specified ECB data level or DECB is verified with the RCC in the record. This check is not made if the specified RCC is zero. If either check fails, an error code is indicated in the ECB. (See “WAITC—Suspend Processing for ECB I/O Completion” on page 554 for more information.)
- To ensure that the operation was completed, a WAITC macro must be run. After a WAITC macro, the CBRW at the specified ECB data level or DECB contains the storage address of the record.
- You cannot call the FINDC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record x
  3. TXSPC
  4. FINDC record x.
- TPF transaction services processing affects FINDC macro processing in the following ways:
  - If a system error occurs because of one of the previous considerations, processing ends as if a TXRBC macro was called. The DASD surface remains unchanged.
  - Finds from general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
  - The TPF system first searches in the commit scope set for the record. If the record is not found, normal DASD retrieval takes place from VFA or the DASD surface.
- 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

None.

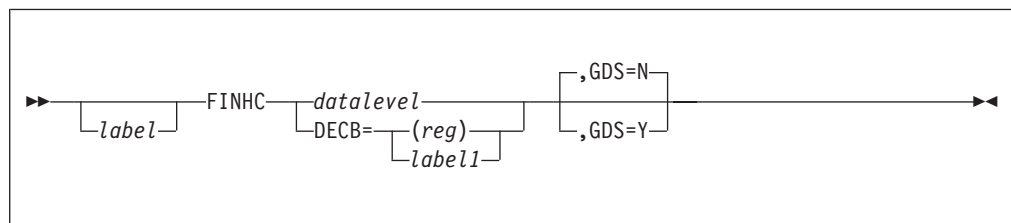
## FINHC—Find and Hold a File Record

This general macro assigns the entry control block (ECB) exclusive control of the specified record and then reads the record from the file into storage. A block of storage is obtained by the control program, and reference to it is stored in the core block reference word (CBRW) at the specified ECB data level or data event control block (DECB).

The requested record can be contained in the virtual file access (VFA).

The control program will queue this request if the record is being held by another entry. All succeeding requests to hold this record are queued until the record is unheld.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

#### **DECB=(reg)||label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

#### **GDS**

Specify one of the following:

- N** The file address in the specified ECB data level or DECB is for a record from the online database. N is the default.
- Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

### Entry Requirements

- 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 (*datalevel*) or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the FARW for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be executed to set up the file address before FINHC is used to access a general data set record.
- The entry must not be holding the record.

### 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.
- 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 status of the operation is unknown.
- The contents of the CBRW is unknown.
- The file address reference word (FARW) at the specified ECB data level (*datalevel*) or DECB is unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is not holding a block of storage at the specified ECB data level or DECB and the file address contained at the specified ECB data level or DECB is valid. If either condition is violated, control is transferred to the system error routine. In addition, the control program verifies that the record ID at the specified ECB data level or DECB is equal to the record ID in the record. If the record ID specified is zero, this check is not made. The RCC at the specified ECB data level or DECB is checked with the RCC in the record. This check is not made if the specified RCC is zero. If either check fails, an error code is indicated in the ECB. (See “WAITC—Suspend Processing for ECB I/O Completion” on page 554 for more information.)
- To ensure that the operation is completed, a WAITC macro must be run. After a WAITC macro, the CBRW at the specified ECB data level or DECB contains the address of the record and the record is held.
- You cannot call the FINHC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record *x*
  3. TXSPC
  4. FINHC record *x*.
- TPF transaction services processing affects FINHC macro processing in the following ways:
  - Additional checks must be made to determine if the file address is held at the program level or at the commit scope level. Requests for file addresses held at the program level are queued as usual. Requests for file addresses held outside of the commit scope are queued. Requests for file addresses held within the commit scope are serviced.
  - If a system error occurs because of one of the previous considerations, processing ends as if a TXRBC macro was called.
  - Finds from general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
  - The TPF system first searches in the commit scope set for the record. If the record is not found, normal DASD retrieval takes place from VFA or the DASD surface.
  - If a deadlock condition is detected on the ECB that issued this macro, a deadlock user exit is called with the ECB address and input/output block (IOB) address as an input. If the return code from the user exit is 0, processing continues as if the user exit was never called. If the return code from the user exit is 4, the ECB is scheduled to exit with dump D9. If the return code from

## FINHC

the user exit is 8, CE1SUD or IDECSUD, and CE1SUG of the ECB will be set to CJCSUHRD and CJCSUDLK (that is, X'81') and the waiting IOB is removed from the system.

- 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

None.

---

## FINIS—Finish Program Assembly

This general macro provides the information necessary to complete assembly of an ECB-controlled (E-type) program. Used with the BEGIN macro to define an E-type program, it defines the system labels necessary to refer to system control blocks and services.

### Format



```
»—FINIS—prog—«
```

*prog*

An optional positional parameter used to name the ECB program. No verification is done on the name specified.

### Entry Requirements

None.

### Return Conditions

None.

### Programming Considerations

- This macro must be placed at the end of an assembler language E-type program.
- This macro calls CPSEQ and REGEQ (REGEQ1 for IBM programs).

### Examples

None.

## FINSC—Find a Single File Record

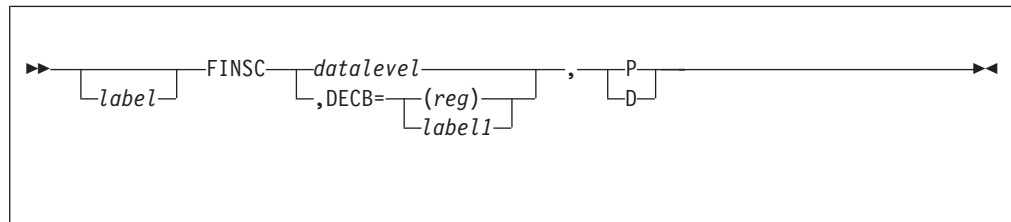
This general macro reads a record from a file into storage. An option provides the user with the facility to read either the primary or duplicate record.

This macro should only be used to read records from the online database. It should not be used to access records on general files or general data sets.

A block of storage is obtained by the control program and reference to it is stored in the core block reference word (CBRW) at the specified entry control block (ECB) data level or data event control block (DECB).

FINSC causes 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.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

**P** Find primary record.

**D** Find duplicate record.

### Entry Requirements

- 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 (*datalevel*) or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.

### 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.
- 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 status of the operation 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 (*datalevel*) or DECB is unchanged.

## Programming Considerations

- This macro can be executed on any I-stream.
- A check is made by the control program to determine if the ECB is not holding a block of storage at the specified ECB data level or DECB, the file address at the specified ECB data level or DECB is valid, and a duplicate exists when it is requested. If either condition is violated, control is transferred to the system error routine. In addition, the control program verifies that the record ID at the specified ECB data level or DECB is equal to the record ID in the record. If the record ID specified is zero this check is not made. The RCC at the specified ECB data level or DECB is checked with the RCC in the record. This check is not made if the specified RCC is zero. If either check fails, an error code is indicated in the ECB (see “WAITC—Suspend Processing for ECB I/O Completion” on page 554 and “Control Program (CP) Detected Errors” on page 22).
- To ensure completion of the operation, a WAITC macro must be run. After a WAITC macro is completed, the CBRW at the specified ECB data level or DECB contains the core address of the record.
- In the event of a noncorrectable error, the Control Program will not service the request from the alternate device (for example, a request for duplicate failed, primary will not be tried).
- 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

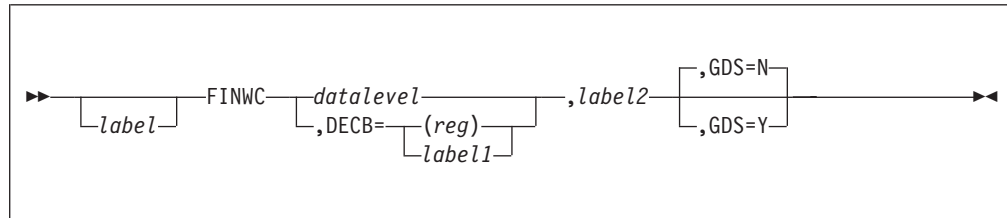
None.

## FINWC—Find a File Record and Wait

This general macro reads a record from a file into storage and waits for completion. A block of storage is obtained by the control program and reference to it is stored in the core block reference word (CBRW) at the specified entry control block (ECB) data level or data event control block (DECB).

The requested record can be contained in the virtual file access (VFA).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

#### **DECB=(reg)||*label1***

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

#### *label2*

The label of an operational program error routine within the current program segment must be specified.

#### **GDS**

Specify one of the following:

- N** The file address in the specified ECB data level or DECB is for a record from the online database.
- Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

### Entry Requirements

- 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 (*datalevel*) or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be executed to set up the file address before FINWC is used to access a general data set record.

### Return Conditions

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The condition code that is returned from this macro cannot be predicted.



- If no I/O hardware errors or unusual conditions have occurred, control is returned to the next sequential instruction. Otherwise, control is returned to the address specified by the variable *label2*.
- All pending input operations, including this request, are completed for this ECB. The status of output requests (except the FILNC macro) is unknown.
- The core block reference word (CBRW) specified by *datalevel* or DECB contains the address of the record.
- The FARW at the specified ECB data level (*datalevel*) or DECB is unchanged.
- For I/O hardware errors, the system error routine has taken a storage dump and informed CRAS.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made by the control program to determine if the ECB is not holding a block of storage at the specified ECB data level or DECB and the file address contained at the specified ECB data level or DECB is valid. If either condition is violated, control is transferred to the system error routine. In addition, the control program verifies that the record ID at the specified ECB data level or DECB is equal to the record ID in the record. If the record ID specified is zero this check is not made. The RCC at the specified ECB data level or DECB is checked with the RCC in the record. This check is not made if the specified RCC is zero. If either check fails, control is transferred to the specified operational program error routine (*label2*).
- An immediate return is made to the user program if all input/output operations are completed at macro time.
- Following use of this macro, control may be transferred to the current program segment for processing of another entry.
- The execution of this macro will reset the 500-millisecond program timeout.
- If any I/O hardware errors or unusual conditions have occurred, detailed information concerning the error has been stored in the ECB for each I/O level and DECB.
- You cannot call the FINWC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record *x*
  3. TXSPC
  4. FINWC record *x*.
- TPF transaction services processing affects FINWC macro processing in the following ways:
  - If a system error occurs because of one of the previous considerations, processing ends as if a TXRBC macro was called.
  - Finds from general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
  - The TPF system first searches in the commit scope set for the record. If the record is not found, normal DASD retrieval takes place from VFA or the DASD surface.
- 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.

**FINWC**

## **Examples**

None.

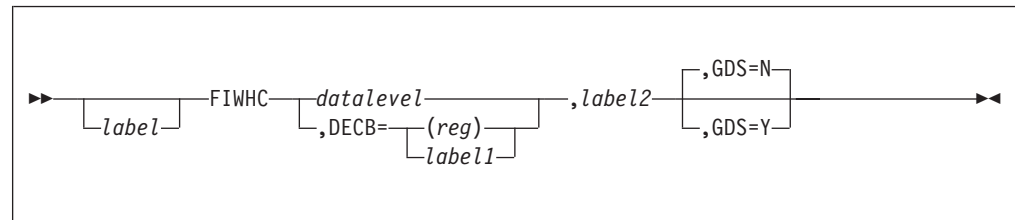
## FIWHC—Find and Hold a File Record, and Wait

This general macro reads a record from file into storage and holds it. A block of storage is obtained by the control program and reference to it is stored in the core block reference word (CBRW) at the specified entry control block (ECB) data level or data event control block (DECB).

The requested record may be contained in virtual file access (VFA).

The control program will queue this request if the record is being held by another ECB. All succeeding requests to hold this record are queued until the existing record hold is released.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

An ECB data level (D0–DF) that identifies the file address and core block information for the I/O request.

#### **DECB=(reg)**/*label1*

The label or general register (R0–R7) containing the address of the DECB, which specifies the file address and core block information for the I/O request.

#### *label2*

The label of an operational program error routine within the current program segment must be specified.

#### **GDS**

Specify one of the following:

**N** The file address in the specified ECB data level or DECB is for a record from the online database.

**Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

### Entry Requirements

- 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 (*datalevel*) or DECB.
- A file address, record ID, and record code check (RCC) must be contained in the file address reference word (FARW) for the specified ECB data level (*datalevel*) or DECB.
- The GDSRC macro must be executed to set up the file address before FIWHC is used to access a general data set record.
- The entry must not be holding the record.

## Return Conditions

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The condition code that is returned from this macro cannot be predicted.
- If no I/O hardware errors or unusual conditions have occurred, control is returned to the next sequential instruction (NSI). Otherwise, control is returned to the address specified by the variable *label2*.
- All pending input operations, including this request, are completed for this ECB. The status of output requests (except the FILNC macro) is unknown.
- The CBRW specified by *datalevel* or DECB contains the address of the record.
- The requested file record is held. (It is available only to this ECB until released).
- The FARW at the specified level (*datalevel*) or DECB is unchanged.
- For I/O hardware errors, the system error routine has taken a core dump and informed CRAS.

## Programming Considerations

- This macro can be executed on any I-stream.
- A check is made by the control program to determine if the ECB is not holding a block of storage at the specified ECB data level or DECB and the file address contained at the specified ECB data level or DECB is valid. If either condition is violated, control is transferred to the system error routine. In addition, the control program verifies that the record ID at the specified ECB data level or DECB is equal to the record ID in the record. If the record ID specified is zero, this check is not made. The RCC at the specified ECB data level or DECB is checked with the RCC in the record. This check is not made if the specified RCC is zero. If either check fails, control is transferred to the specified operational program error routine (*label2*).
- An immediate return is made to the user program if all input/output operations are completed at macro time.
- Following use of this macro, control may be transferred to the current program segment for processing of another entry.
- Running this macro resets the 500-millisecond program time-out.
- If any I/O hardware errors or unusual conditions have occurred, detailed information concerning the error has been stored in the ECB for each I/O level and DECB.
- You cannot call the FIWHC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record *x*
  3. TXSPC
  4. FIWHC record *x*.
- TPF transaction services processing affects FIWHC macro processing in the following ways:
  - Additional checks must be made to determine if the file address is held at the program level or at the commit scope level. Requests for file addresses held at the program level are queued as usual. Requests for file addresses held outside of the commit scope are queued. Requests for file addresses held within the commit scope are serviced.
  - If a system error occurs because of one of the previous considerations, processing ends as if a TXRBC macro was called.

- Finds from general files or general data sets are not considered part of the commit scope and are not affected by commit scope processing.
- The TPF system first searches in the commit scope set for the record. If the record is not found, normal DASD retrieval takes place from VFA or the DASD surface.
- If a deadlock condition is detected on the ECB that issued this macro, a deadlock user exit is called with the ECB address and input/output block (IOB) address as an input. If the return code from the user exit is 0, processing continues as if the user exit was never called. If the return code from the user exit is 4, the ECB is scheduled to exit with system error dump CTL-0D9. If the return code from the user exit is 8, CE1SUD or IDECSUD, and CE1SUG of the ECB will be set to CJCSUHRD and CJCSUDLK (that is, X'81') and the waiting IOB is removed from the system.
- 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

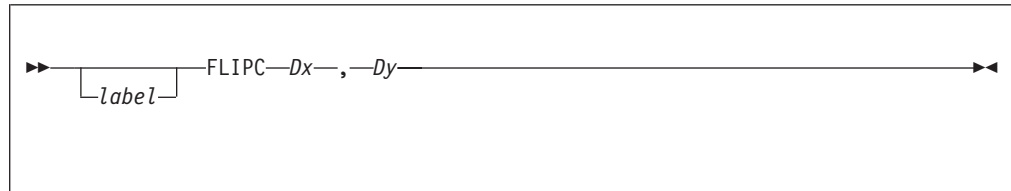
None.

## FLIPC—Interchange the Status of Two Data Levels

This general macro causes the data contained in the ECB control fields associated with the 2 specified data levels to be interchanged.

The file address reference words (FARW), file address reference word extensions, core block reference words (CBRW), and detail error indicator fields are interchanged.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* The symbolic index (D0-DF) of the first ECB data level to be interchanged with the second.

*Dy* The symbolic index (D0-DF) of the second ECB data level to be interchanged with the first.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- Both data levels specified in the FLIPC macro must not have any I/O operations in progress where the status of the storage block is not known until a subsequent WAITC macro is invoked.

### 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 condition code will be saved across the macro call when the executing in 24-bit addressing mode.
- If either or both of the specified ECB data levels are holding a block of storage prior to macro execution, the block (or blocks) will be held by the opposite request level following execution of this macro.
- The core storage reference words for the specified ECB data levels have been interchanged.
- The FARW for the specified levels have been interchanged.
- The FARW extensions for the specified levels have been interchanged.
- The detail error indicators for the specified levels have been interchanged.

### Programming Considerations

- This macro can be executed on any I-stream.
- If the same level is specified for both data levels, the macro expansion will result in a “NOP” instruction.

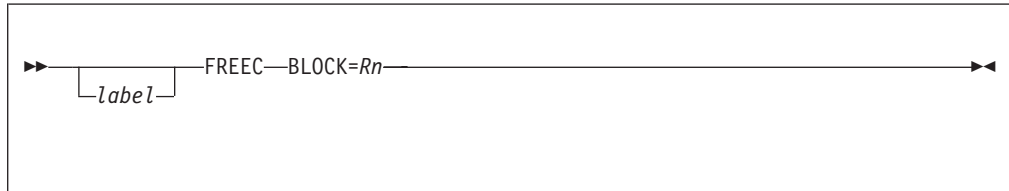
## Examples

None.

## FREEC—Release Storage Blocks

This general macro returns to the heap a block of storage allocated from the heap by a MALOC, CALOC, or RALOC macro call.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **BLOCK=Rn**

A general register (R0-R7, R14, R15) or a register equate containing the address of a heap-resident storage block no longer in use by an application and available for reallocation. The storage block must previously have been allocated by a MALOC, CALOC, or RALOC call.

When you code this parameter in the control program (CP), you can specify only R0-R6.

### Entry Requirements

- The general register indicated by BLOCK must contain the base address of a block of storage allocated using the MALOC, CALOC, or RALOC macros (or their C language functional equivalents).
- When called from the CP:
  - The caller must be in the ECB virtual memory (EVM).
  - The caller must be in 31-bit addressing mode.
  - R10 cannot be used as the base register.
  - R9 must point to the entry control block (ECB).

### Return Conditions

- The contents of the register specified by BLOCK are unpredictable and should be reloaded with another value before reuse.
- The contents of R14 and R15 are unknown. All other registers are preserved.
- When called from the CP, the contents of R7 are not preserved across this macro call.
- Control is not returned if corruption is detected in the heap. A system error occurs, and the ECB exits immediately. See *Messages (System Error and Offline)* and *Messages (Online)* for more details about this error.
- Control is returned to the NSI if no heap corruption is detected.

### Programming Considerations

- The general register specified by BLOCK must contain the address of a block of storage returned from the last MALOC, CALOC, or RALOC operation against it. Otherwise, this function may conclude that there is corruption in the heap, causing the immediate exit of the ECB, unless the address is zero.
- This macro can be executed on any I-stream.



- In addition to the normal macro trace information the macro trace for this macro contains the address of the storage block freed.
- When called from the CP, the contents of the ECB register save area (CE1S0x) are not preserved. For example, for BLOCK=5, CE1S05 is not preserved.

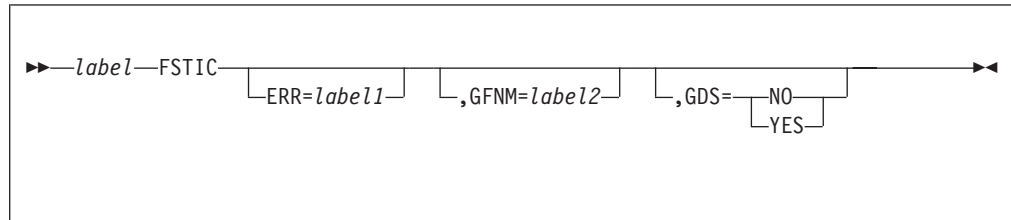
## Examples

- Return a storage block pointed to by R4 to heap storage.  
FREEC BLOCK=R4

## FSTIC–File Status Table Information

This general macro accepts a symbolic module number from the requester and responds with information about that symbolic module taken from the appropriate disk status table. The information is returned in a consistent format, even though the individual status tables are not in a consistent format.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ERR=label1**

An optional parameter for a label to which the program will branch if the symbolic module number is illegal.

#### **GFNM=label2**

An optional parameter for a label to which the program will branch if the symbolic module number refers to a “pseudo mod”, a general file that is not mounted.

#### **GDS**

An optional parameter that specifies whether general data set module numbers are to be processed.

#### **NO**

Any symbolic module number that is greater than the COPY MOD number is considered invalid.

#### **YES**

Any symbolic module number in the number range for general data set modules is processed as valid.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R15 must contain, in binary, the symbolic module number for which information is desired.

## Return Conditions

- Control is returned to the next sequential instruction.
- 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.
- If the supplied symbolic module number is illegal, R15 will contain 0's.
- If the supplied symbolic module number is legal, then:

R14 bytes 0–1 = Number of duplicate module  
 2–3 = Hardware address of module  
 R15 bytes 0–1 = Unit status, as follows:

| Bit(s) If off: |                         | If on:                 |
|----------------|-------------------------|------------------------|
| Byte 0:        |                         |                        |
| 0              | Normal                  | Reserved               |
| 1              | Normal                  | Reserved               |
| 2              | Normal                  | Reserved               |
| 3              | Normal                  | Copy State             |
| 4              | Normal                  | Reserved               |
| 5              | Online                  | Offline                |
| 6              | Non-dup'd module        | Module is dup'd        |
| 7              | Wholly dup'd            | Partially dup'd        |
| Byte 1:        |                         |                        |
| 0              | CKD CCWs                | ECKD (*) CCWs          |
| 1              | Real-time module        | General module         |
| 2              | Normal                  | Copy module            |
| 3              | Normal                  | General Dataset module |
| 4              | Normal                  | Cache CU               |
| 5              | Normal                  | RCS CU                 |
| 6              | Normal                  | CU supports prefixing  |
| 7              | Reserved for future use |                        |
| Byte 2:        |                         |                        |
|                | DASD                    |                        |
|                | X'00' = DEVA            |                        |
|                | X'04' = DEVB            |                        |
|                | X'08' = DEVC            |                        |
|                | X'0C' = DEVD            |                        |
| Byte 3:        |                         |                        |
|                | 0001xxxx                |                        |
|                | Bit 3 = 1               | DASD                   |
|                | Bits 4–7 = xxxx         |                        |

#### Device Type

Where:

|           |
|-----------|
| 0110–3350 |
| 1000–3375 |
| 1010–3380 |
| 1100–3390 |
| 1101–9345 |

## Programming Considerations

- This macro can be executed on any I-stream.
- It is the user's responsibility to know which bits of returned information do not apply to the particular device type and to ignore such information.

## Examples

None.

## GCFLC–Get Core Block and Large File Address.

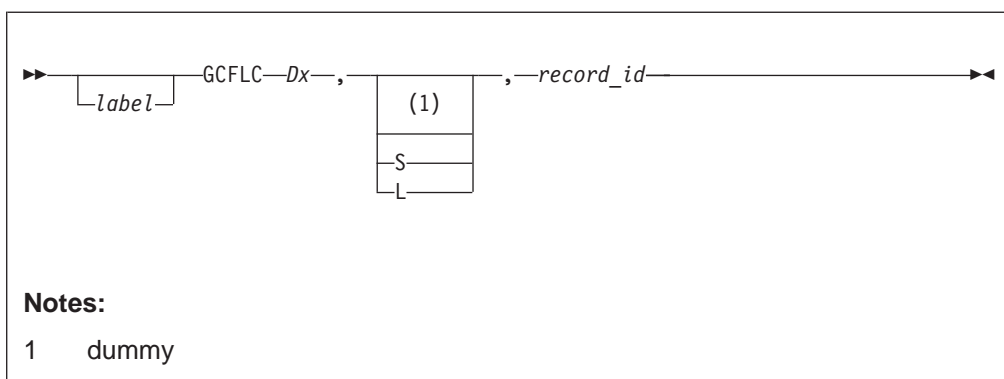
This general macro is a combination of the control program macros GETLC and GETCC. This macro obtains an available large block of storage and an available large file storage address from the short or long term pool records indicated by the requested record id.

The address and size of the storage block is placed in the specified core block reference word (CBRW) of the specified ECB data level.

The address of the file record is inserted into the file address reference word (FARW) of the specified ECB data level.

**Note:** This macro is provided for backward compatibility only. New programs should be written using the GETFC macro (see “GETFC–Get File Pool Address and Storage Block” on page 248).

## Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* This parameter must be coded as an ECB data level D0-DF.

*dummy*

This is a dummy parameter that no longer has meaning. It is required to preserve the position of the parameter following. The value S or L can be used, or it may just be indicated with commas.

*record\_id*

A valid record ID, the characteristics of which are used to determine whether a long term, short term, or duplicate file address is obtained.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The ECB must not be holding a storage block for the specified data level.

## Return Conditions

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

## Programming Considerations

- This macro can be executed on any I-stream.
- Only one block of storage may be obtained at a time through this macro. Separate requests must be made for each block.
- Only one record address may be obtained at a time through this macro. Separate requests must be made for each record address.
- A check is made by the control program to determine if the ECB requesting a storage block is already holding one for the specified data level. If a block is already held, control is transferred to the system error routine.
- Core blocks held by an operating program should be released as soon as possible by the operating program.
- All file storage record addresses must ultimately be released by a RELFC macro.
- No record address may be released twice.
- This macro may result in the equivalent of an implied WAITC.
- File pool addresses that are acquired in a global transaction (that is, after a TXBGC macro call but before a TXCMC or TXRBC macro call) are released if the transaction is rolled back using a TXRBC macro call. However, the core block remains attached to the ECB.

## Examples

GCFLC D0,L,OM

The address and size of a storage block of type OM is put into the core block reference word for data level D0. Whether the block is from the long term pool or the short term pool depends on the definition of the OM record returned. The L is merely a placeholder and has no other meaning.

## GCFSC—Get Small Core Block and File Address

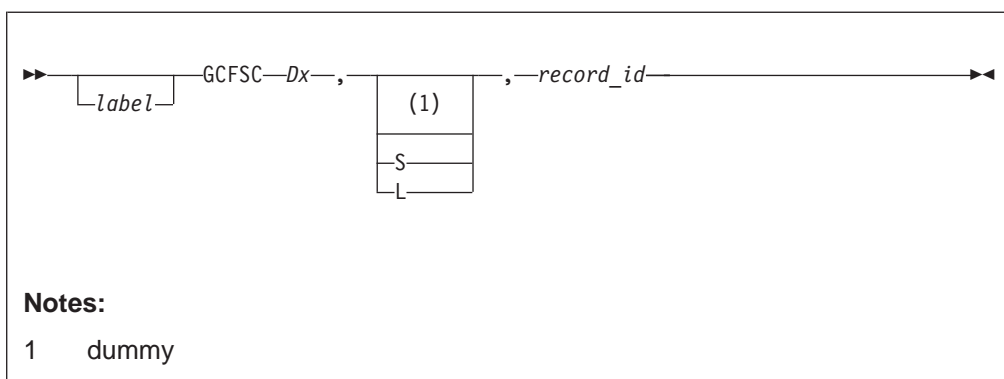
This general macro is a combination of the control program macros GETSC and GETCC. This macro obtains an available block of small storage and an available small file storage address from the short or long term pool of small storage records indicated by the specified record id.

The address and size of the storage block is placed in the core block reference word (CBRW) of the specified entry control block (ECB) data level.

The address of the file record is inserted into the file address reference word (FARW) of the specified ECB data level.

**Note:** This macro is provided for backward compatibility only. New programs should be written using the GETFC macro (see “GETFC—Get File Pool Address and Storage Block” on page 248).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* This parameter is a valid ECB level D0 thru DF.

*dummy*

This parameter no longer has meaning. It is required to preserve the position of the parameter following. It can be coded as S or L, or indicated with commas.

*record\_id*

A valid record ID whose characteristics will be used to determine whether a long term, short term or duplicate record address is returned.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The ECB must not be holding a core block on the specified data level.

### Return Conditions

- Control is returned to the next sequential instruction.
- The content of R15 is unknown.
- R14 contains the address of the core block.
- The address and size of the block assigned is placed in the CBRW for the specified data level.

- A record address is placed into the specified file address reference word.
- The contents of the remaining registers and the condition code are preserved across this macro.

## Programming Considerations

- This macro can be executed on any I-stream.
- Only one block of storage may be obtained at a time through this macro. Separate requests must be made for each block.
- Only one record address may be obtained at a time through this macro. Separate requests must be made for each record address.
- A check is made by the control program to determine if the ECB requesting a block is already holding one for the specified data level. If a block is already held, control is transferred to the system error routine.
- Storage blocks held by an operational program should be released as soon as possible by the operational program.
- All file storage record addresses must ultimately be released by a RELFC macro.
- No record address may be released twice.
- The GETSC macro may result in the equivalent of a WAITC.
- File pool addresses that are acquired in a global transaction (that is, after a TXBGC macro call but before a TXCMC or TXRBC macro call) are released if the transaction is rolled back using a TXRBC macro call. However, the core block remains attached to the ECB.

## Examples

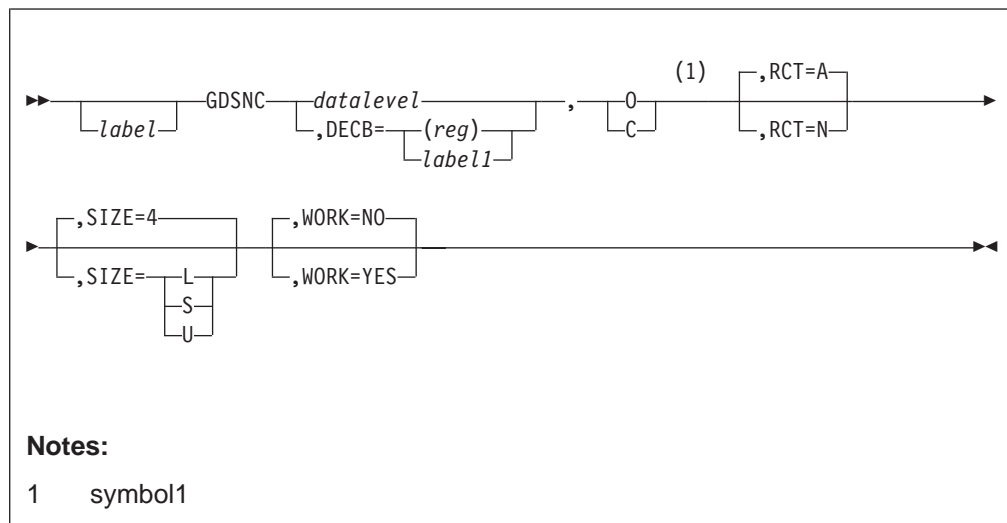
GCFSC D0,L,OM

The address and size of a storage block of type OM is put into the core block reference word for data level D0. Whether the block is from the long term pool or the short term pool depends on the definition of the OM record returned. The L is merely a placeholder and has no other meaning.

## GDSNC—Get General Data Set Entry

This general macro is provided to initialize a user's file address reference word (FARW) with the data necessary to access a general data set or a volume of a general data set.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

This is a core block reference word (CBRW); file address reference word (FARW); and file address extension word (FAXW) (D0–DF).

#### **DECB=(reg)|*label1***

The label or general register (R0–R7) containing the address of the data event control block (DECB), which identifies the CBRW, FARW, and FAXW.

#### *symbol1*

This is an open (O) or close (C) character.

#### **RCT**

This is the record number format of the data set; the default value is A. Valid values are:

**A**   TPF record number format

**N**   Non-TPF (OS/VS) record number format

#### **SIZE**

The record size of the data set. The default value is 4. Valid values are:

**L**   Large (1055 bytes)

**S**   Small (381 bytes)

**U**   Undefined

**4**   4K (4095 bytes)

The SIZE = U parameter cannot be used with TPF data sets. For TPF data sets only SIZE = L, S, or 4 will be accepted.



The SIZE = U parameter is used for non-TPF data sets; such as OS data sets. When SIZE = U is specified the data set is divided into 4K-byte allocations. If the data set was not defined by the user to 4K-byte allocations, any additional records at the end of the tracks will not be addressable.

#### WORK

The location where the relative record number (RRN) of the desired record in the data set is specified. The default is NO. Valid values are:

#### NO

Relative record number is in the FARW extension.

#### YES

Relative record number is in the user's work area.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must contain the address of a 16-byte field. This field must contain a 1-16 character data definition name. If the data definition name is less than 16 bytes, then it must be left-justified within this field and the remaining bytes padded with blanks.
- The CE1FMx field for the ECB data level or the IDECFM field of the DECB specified is initialized to zero or the volume sequence number desired.
- When the WORK=NO parameter is specified, the first fullword of the CE1FXx field of the ECB data level or the IDECFX0 field of the DECB must contain zeros or the RRN of a desired record in the data set.
- When the WORK=YES parameter is specified, a 4-byte field immediately following the data set name field must contain zeros or the relative record number of a desired record within the data set.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R15 are unknown. Contents of all other registers are preserved across this macro call.
- Normal Return
  - R14 is set to zero.
  - The CE1FMx, CE1FCx, CE1FHx and CE1FRx fields for the ECB data level or the IDECFM, IDECFC, IDECFH and IDECFR fields for the DECB specified will contain the indexes necessary for the TPF system to access the records of the data set. These fields must not be changed by the user program until this data set is closed by this ECB.
  - The CE1FXx field for the ECB data level or the IDECFX0 field of the DECB specified contains a file address of the desired record in the data set.

**Note:** When the RRN requested is zero, the address returned in field CE1FXx or field IDECFX0 is the address of the first record of the data set.

- Error Return
  - R14 is set to an error return code. Depending upon the error condition the fields in the ECB or DECB may or may not be changed. Error return codes are:
    - 4 The RRN is not valid (data set). ECB fields CE1FMx, CE1FCx, CE1FHx, CE1FRx for the specified ECB data level or fields IDECFM, IDECFC,

## GDSNC

IDECFH and IDECFR of the DECB have been initialized. Field CE1FXx for this ECB data level or field IDECFX0 of the DECB remain unchanged.

- 8** The RRN is not valid (volume). ECB or DECB fields are unpredictable.
- 12** The volume sequence number is missing. ECB or DECB fields are unpredictable.
- 16** The requested DD name (DDNAME) was not mounted. ECB or DECB fields are unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- Except for the record ID and RCC portions of the FARW, the data inserted into the FARW by the control program must not be altered.

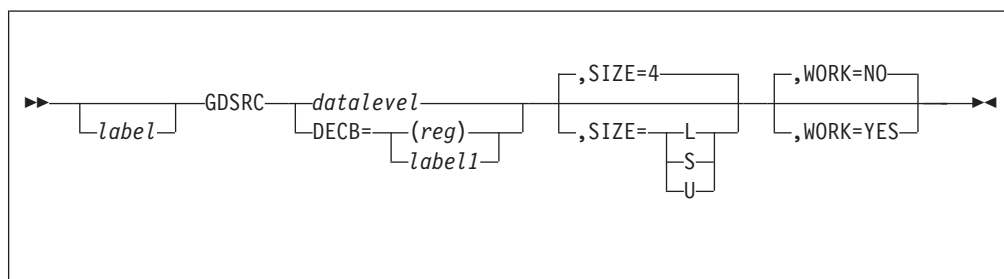
## Examples

None.

## GDSRC—Get General Data Set Record

This general macro is provided to convert a relative record number (RRN) into a file address (CCHR format) to access a general data set record.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

A file address reference word (FARW) and file address extension word (FAXW) (D0–DF).

#### **DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB) that identifies the FARW and FAXW.

#### **SIZE**

The record size of the data set. The default value is 4. Valid values are:

- L** Large (1055 bytes)
- S** Small (381 bytes)
- U** Undefined
- 4** 4KB (4095 bytes).

The SIZE = U parameter is used for non-TPF data sets, such as OS data sets. When SIZE = U is specified, the data sets are divided into 4KB allocations. If the data set was not defined by the user to 4KB allocations, any additional records at the end of the tracks will not be addressable.

#### **WORK**

The location where the RRN of the desired record within the data set is specified. The default is NO.

##### **NO**

The RRN is in the FARW extension.

##### **YES**

The RRN is in the user's work area.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must contain the address of a 16-byte field. This field must contain a 1–16 character data definition name. If the data definition name is less than 16 bytes, then it must be left-justified within this field and the remaining bytes padded with blanks.

## GDSRC

- When the WORK=NO parameter is specified, field CE1FXx of the ECB data level or field IDECFX0 of the DECB must contain the RRN of a desired record in the data set.
- When the WORK=YES parameter is specified, a 4-byte field immediately following the data set name field must contain the RRN of a desired record.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R15 are unknown. The contents of all other registers are preserved across this macro call.
- Normal Return
  - The contents of R14 are set to zero.
  - Field CE1FXx of the ECB data level or field IDECFX0 of the DECB specified will contain a file address (in CCHR format) of the desired record in the data set as specified by the user via the WORK parameter.

**Note:** When the RRN specified is zero, the address returned in field CE1FXx or field IDECFX0 is the address of the first record of the data set.

- Error Return
  - R14 is set to an error return code. Depending on the error condition, the fields in the ECB or DECB may or may not be changed. Error return codes are:
    - 4** The RRN is not valid (data set). The CE1FXx field for this ECB data level or the IDECFX0 field of the DECB remains unchanged.
    - 8** The RRN is not valid (volume). The CE1FXx field for this ECB data level or the IDECFX0 field of the DECB remains unchanged.
    - 12** This is the missing volume sequence number. ECB or DECB fields are unpredictable.
    - 16** The requested data set was not mounted. ECB or DECB fields are unchanged.
    - 24** The Device Dependent Table was not found. ECB or DECB fields are unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- Except for the record ID and RCC portions of the FARW, the data inserted into the FARW by the control program must not be altered.

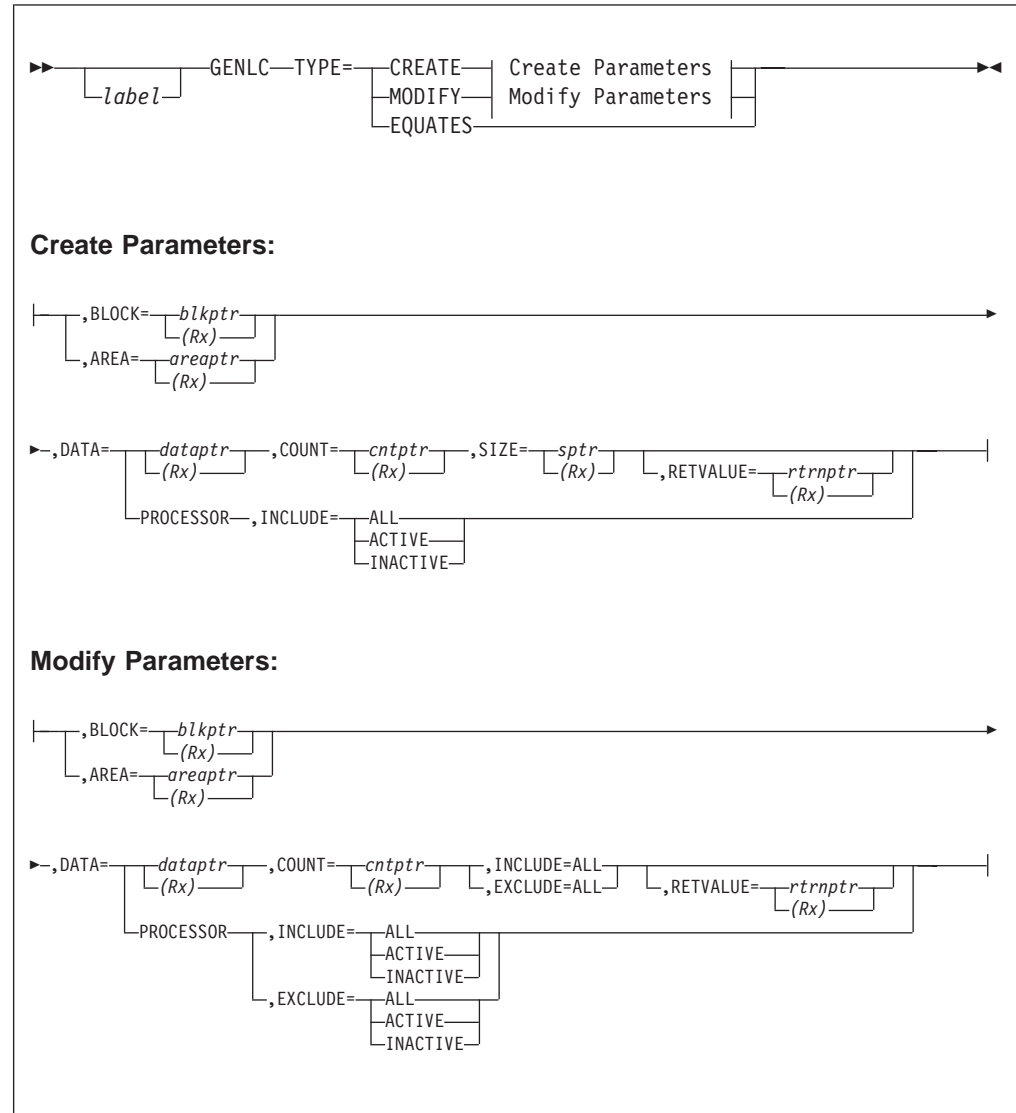
## Examples

None.

## GENLC—Generate a Data List

Use this general macro to generate a list of data items that can be used with SIPCC and event-type macros. Data item lists are used to define a variable list of items associated with a particular event or set of interprocessor communications (IPC) processor destinations.

### Format



*label*

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

#### TYPE

specifies the type of operation requested, where:

##### CREATE

creates a new data item list.

##### MODIFY

modifies an existing data item list.

### EQUATES

creates DSECT labels for the list definition fields.

### BLOCK

specifies that the data item list is to be built or modified in an event block (defined by the EV0BK macro), where:

*blkptr*

is a label specifying the address of the event block.

*Rx* is a register (R0 to R13, or R15) containing the address of the event block.

### AREA

specifies that the data item list is to be built or modified in a work area, where:

*areaptr*

is a label specifying the address where the data item list begins in the work area.

*Rx* is a register (R0 to R13, or R15) containing the address where the data item list begins in the work area.

### DATA

specifies the type of data or the location of data that will be included or excluded from the list, where:

*dataptr*

is a label specifying the address where data items can be found.

#### Notes:

1. For TYPE=CREATE, both the COUNT and the SIZE parameters must be specified.
2. For TYPE=MODIFY, the COUNT parameter must be specified.

*Rx* is a register (R0 to R13, or R15) containing the address where the data items can be found.

### COUNT

specifies the count of data items in the data item list, where:

*cntptr*

specifies the address of a halfword field that contains the count.

*Rx* specifies a register (R0 to R13, or R15) that contains the count.

### SIZE

specifies the size, in bytes, of a data item, where:

*sptr*

specifies the address of a halfword field that contains the size.

*Rx* specifies a register (R0 to R13, or R15) that contains the size.

### RETVALUE

specifies a return value to be loaded into the return value field of the data items.

*rtrnptr*

specifies the address of a halfword field that contains the count.

*Rx* specifies a register (R0 to R13, or R15) that contains the count.

#### Notes:

1. The return value is stored in the number of data items specified in the COUNT parameter.

2. If RETVALUE is not specified, data item return values default to zero.

**PROCESSOR**

specifies that the data item list contains processor ordinal numbers as defined in the processor ID table (PIDT).

**Note:** For DATA=PROCESSOR, the INCLUDE or EXCLUDE (MODIFY only) parameter must be specified.

**INCLUDE**

specifies which data items to include in the list or to add to the list.

**Notes:**

1. When TYPE=CREATE, INCLUDE is required if DATA=PROCESSOR and is not valid if DATA points to an area containing data items.
2. When TYPE=MODIFY, either INCLUDE or EXCLUDE must be specified.

**ALL**

specifies that all the data items in the data item list or the ordinal numbers for all the processors are to be included or added to the data item list.

**ACTIVE**

specifies that the ordinal numbers for the active processors are to be included or added to the data item list.

**Notes:**

1. The ACTIVE parameter is valid only when DATA=PROCESSOR is specified.
2. The originating processor is not included or added to the list.

**INACTIVE**

specifies that the ordinal numbers of the inactive processors are to be included or added to the data item list.

**Notes:**

1. The INACTIVE parameter is valid only when DATA=PROCESSOR is specified.
2. The originating processor is not included or added to the list.

**EXCLUDE**

specifies which data items or processor ordinals are **not** included in the list.

**Note:** EXCLUDE is valid only when TYPE=MODIFY.

**ALL**

specifies that all the indicated data items or all the processor ordinals are to be removed from the data item list.

**ACTIVE**

specifies that the ordinal numbers of the processors that are currently active are to be removed from the list.

**Notes:**

1. The ACTIVE parameter is valid only when DATA=PROCESSOR is also specified.
2. The originating processor, if part of the list, is not removed.

**INACTIVE**

specifies that the ordinal numbers of the inactive processors are to be removed from the data item list.

## GENLC

### Notes:

1. The INACTIVE parameter is valid only when DATA=PROCESSOR is also specified.
2. The exclude process does not exclude the originating processor.

## Entry Requirements

- Register 9 (R9) must contain the address of the entry control block (ECB) that is being processed.
- When TYPE=MODIFY is specified, the BLOCK or AREA parameter must specify a list that was previously created.
- The data items pointed to by the DATA parameter must be contiguous elements of the same length as specified by the SIZE parameter. The data item size contained in the generated list will be 6 bytes larger than the size defined by the SIZE parameter.
- For DATA=PROCESSOR, each list data item is a 1-byte processor ordinal number.
- The total data size, where  $\text{total} = \text{COUNT} * (\text{SIZE} + 6)$ , cannot exceed 400 bytes.
- When the GENLC macro is called in the control program, the following conditions exist:
  - The GENLC service routine is run in the ECB virtual memory (EVM). The caller should be in EVM.
  - The GENLC service routine uses ECB work area CE1MAC. The contents of this area are not preserved for the caller.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- If the BLOCK parameter is specified, the generated list is returned in EV0BK at EVNBKLI.
- If the AREA parameter is specified, the generated list is returned at the beginning at the specified area.
- The data list itself consists of the following format in the order shown:
  - A halfword count of list data items
  - A halfword list data item size (the input data size + 6)

**Note:** For processor type data, the returned list item size is 7 bytes.

- The list of data items; each list data item is sorted as follows:
  - A 1-byte flag or status field, defined by use
  - A 1-byte indicator field, also defined by use
  - A 4-byte return value field
  - The input data item as specified with the DATA parameter.
- The contents of R14 and R15 are unknown.

## Programming Considerations

- Do not use register 14 (R14) to pass GENLC parameter addresses or values.
- The GENLC macro uses information contained in the processor ID table (PIDT).
- The EV0BK DSECT describes the user area that will be used as a parameter to the following macros:
  - EVINC



- EVNQC
- EVNTC
- EVNWC
- POSTC
- SAWNC.

## Examples

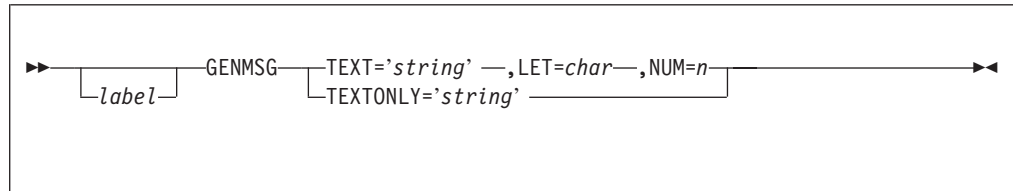
- The following example builds, in the event block, a processor data list consisting of only active processors.

```
GENLC TYPE=CREATE,BLOCK=(R5),DATA=PROCESSOR,INCLUDE=ACTIVE
```

- The following example removes the specified processor from an existing processor data list.

```
GENLC TYPE=MODIFY,AREA=EBX000,DATA=PI1IPT, COUNT=EBW000, EXCLUDE=ALL
```

**Note:** This is a declarative macro. It is used only to generate data.



A symbolic name can be assigned to the macro statement, which can be used to gain addressability to the generated parameter table.

This specifies a literal message severity indicator letter. This is a mandatory parameter when used with TEXT=.

This specifies a self-defining term for the message number assigned to the message in one of two ranges: either decimal 1–255 or 1–9999. The range is dependent on previous use of DCTMSG or WTOPC. If XNUM=YES is specified in DCTMSG or WTOPC, the large range is used. NUM is a mandatory parameter when used with TEXT=.

This specifies the text of the message to be edited. The text must be enclosed in quotes. The minimal length of the text is 1 character.

This specifies the text of the message to be edited and generates a count field followed by the message text. This parameter can be used to generate message text tables. TEXTONLY cannot be addressed with DCTMSG.

**Note:** See the WTOPC macro for a complete definition of the contents of the LET, NUM, and TEXT parameters, and number range.

- LET and NUM must always be coded when TEXT is used.
- When TEXTONLY is coded all other parameters will be ignored.

None.

- “SERRC–System Error” on page 357

- “SNAPC—Snapshot Dump” on page 363
- “WTOPC—Edit and Send System Message” on page 559.

## Examples

- The following example generates entries to be used with the WTOPC macro.

**Note:** See the WTOPC macro for more information about the substitution used in this example.

```

LA R2,MSGTXT1
B SENDMSG
.
.
.
LA R2,MSGTXT2
LA R3,=C'SECOND'
B SENDMSG
.
.
.
DCTMSG
USING DCTMSG,R2
SENDMSG WTOPC PREFIX=PROG, X
 NUMA=MSGNUM, X
 LETA=MSGLET, X
 TEXTA=MSGLEN, X
 SUB=(CHARA,(R3))
DROP R2
.
.
.
MSGTXT1 GENMSG NUM=1,LET=I,TEXT='THIS IS THE FIRST MESSAGE'
MSGTXT2 GENMSG NUM=2,LET=I,TEXT='THIS IS THE MESSAGE'
```

This example produces the following messages:

```

PROG01I 00:00:00 THIS IS THE FIRST MESSAGE
PROG02I 00:00:00 THIS IS THE SECOND MESSAGE
```

- The following example shows how to use the GENMSG macro to generate message text for the SNAPC macro.

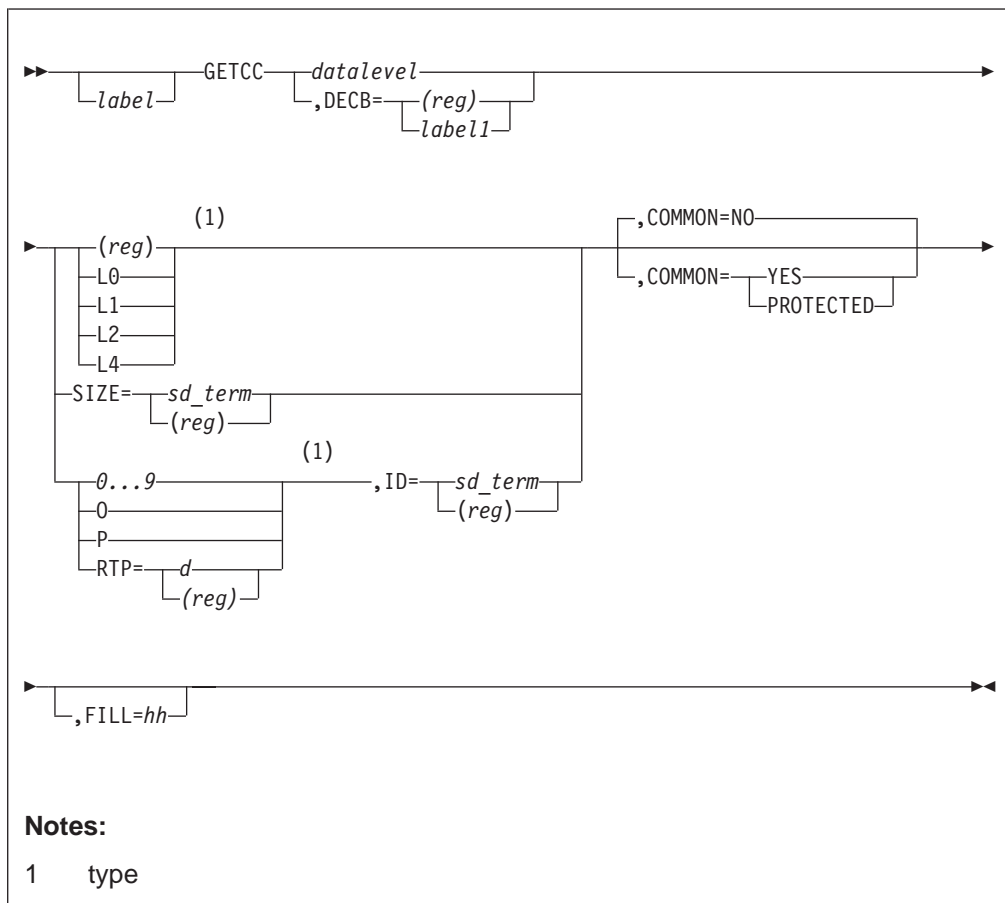
```

SNAPC R,(CE9380),REGS=YES,MSG=ERR_MSG
.
.
ERR_MSG GENMSG TEXTONLY='UNABLE TO RETRIEVE IFMSG TABLE'
```

## GETCC—Get a Working Storage Block

This general macro assigns a storage block of specified size and type to an entry control block (ECB). The address and logical size of the storage block are placed in the specified core block reference word (CBRW) in the ECB.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### *datalevel*

A symbolic data level index value (D0–DF).

#### **DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB).

#### *type*

A symbolic block type or a register in the form (reg) containing the block type may be specified for this parameter. The type and SIZE parameters are mutually exclusive. Block types provided and validated by the TPF system are:

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

Registers R0–R7 can be used for (*reg*).

**Note:** Additional block type equates must be added to the user modifiable macro IVTYPE if they are to be considered valid.

When the ID keyword is coded and the RTP keyword is not used, the second positional parameter is used to define the record ID type. Valid types for this variant are:

0,1,...9

A digit from 0–9, signifying the selection of a combination of size, duration, and device type attributes as defined in the RIATA entry for the record identified by the ID parameter. See the RIATA information in *TPF System Generation* for further information on RTPx.

In previous TPF releases, this parameter was either P, for primary pool type or O for overflow pool type. P and O are still valid for migration and correspond to 0 and 1 respectively.

**Note:** The second positional parameter is not coded when the SIZE keyword is coded.

**ID**=*sd\_term*[(*reg*)

This is a self-defining, 2 character record identity (ID) or a register in the form (*reg*) containing the record identity. An example of a record ID is ID=CAA. The specified record ID must be in the record ID attribute table (RIAT).

**RTP**=*d*[(*reg*)

A digit or register containing a digit from 0 to 9, which indicates that a combination of size-, duration-, and device-type attributes were selected as defined in the RIATA macro entry for the record identified by the ID parameter. Valid registers are R0–R7. See the RIATA macro information in *TPF System Generation* for more information about RTPx.

**SIZE**=*sd\_term*[(*reg*)

This is a self-defining term for the block size (in bytes) or a register in the form (*reg*) containing the block size. An example of a value for block size might be SIZE=350. The largest block size supported is 4095. The self-defining term should be expressed either by using the hexadecimal type of specification or by the decimal numeric constant.

## COMMON

Allocated block storage can be made accessible to all other ECBs or protected from them by using this parameter.

### YES

The storage comes from the pool of shared main storage. Any ECB can access the storage block.

### NO

The storage is unique to this ECB and is protected from reference or alteration by other ECBs. NO is the default.

### PROTECTED

The storage comes from the pool of shared main storage. Any ECB can access the storage block.

**Note:** The usage of the COMMON=YES parameter requires common block authorization (CHECK=CMB) in the IBMPAL macro.

## GETCC

### **FILL=hh**

An optional, 1-byte, hexadecimal value (hh) used to initialize the user section of the storage block being allocated.

For example, by coding FILL=00, a storage block allocated is initialized to X'00's.

If the FILL parameter is not specified, the storage block contents are unpredictable; unless the SYSTC tag SBCLEAR is set on. SBCLEAR set on indicates that the system is to clear all blocks (with zeros) when the blocks are assigned to the ECB. The FILL parameter has priority over the SBCLEAR setting.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The CBRW for the specified ECB data level or DECB must not have a storage block attached.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- R14 contains the address of the assigned storage block.
- The contents of R15 are unknown. The contents of all user registers are preserved across this macro call.
- The address of the assigned storage block is placed in the CBRW for the specified ECB data level or DECB.

## Programming Considerations

- This macro can be run on any I-stream.
- COMMON storage and PROTECTED storage are scarce resources in the TPF system. Their use should be restricted. Only use the COMMON=YES or COMMON=PROTECTED parameters when a storage block must be passed to other ECBs.
- Initializing a main storage block with the FILL option, or through any other method, takes more CPU resource than not initializing the block.
- Only one storage block at a time may be obtained through this macro. Separate requests must be made for each storage block.
- Storage blocks held by a program should be released as soon as possible by the program.
- Only one set of parameters describing the block size can be coded. The SIZE parameter is mutually exclusive with the ID parameter and with the symbolic block type (Lx) parameter.
- The COMMON and FILL keywords can be coded with any of the block size parameters.
- When a register is selected as a parameter it must be in the range R0-R7 and take the form (Rx).
- An example of a self-defining term for the SIZE keyword would be X'17D' or 381, if a storage block large enough to store 381 bytes of data is required.
- If SBCLEAR is set on (with the ZSYSG command), blocks assigned by this macro are initialized to zero. Initializing blocks by this or any other manner will take additional CPU resources.
- When the ID keyword is used, the record ID must be in the RIAT table and the device type attribute coded for TYPE must be defined for the specified ID.

- The IVTYPE macro validates the block type parameter for the GETCC macro. It can be modified to contain user defined block types as valid block type equates for GETCC macro calls. It is called by the GETCC macro only and returns to NSI in GETCC if block type is valid, or to label ERR7 in the GETCC macro call if block type is invalid.

Users can add their block type equates to the list of types in the macro and avoid assembly errors when using non-standard equates. The addition is done by modifying the AIF test for legitimate types:

```
AIF ('&P2' NE 'L0' AND '&P2' NE 'L1' AND '&P2' NE 'L2' AND
 '&P2' NE 'L4').ERR7
```

The types provided by TPF (L0, L1, L2, and L4) should not be changed.

- 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.

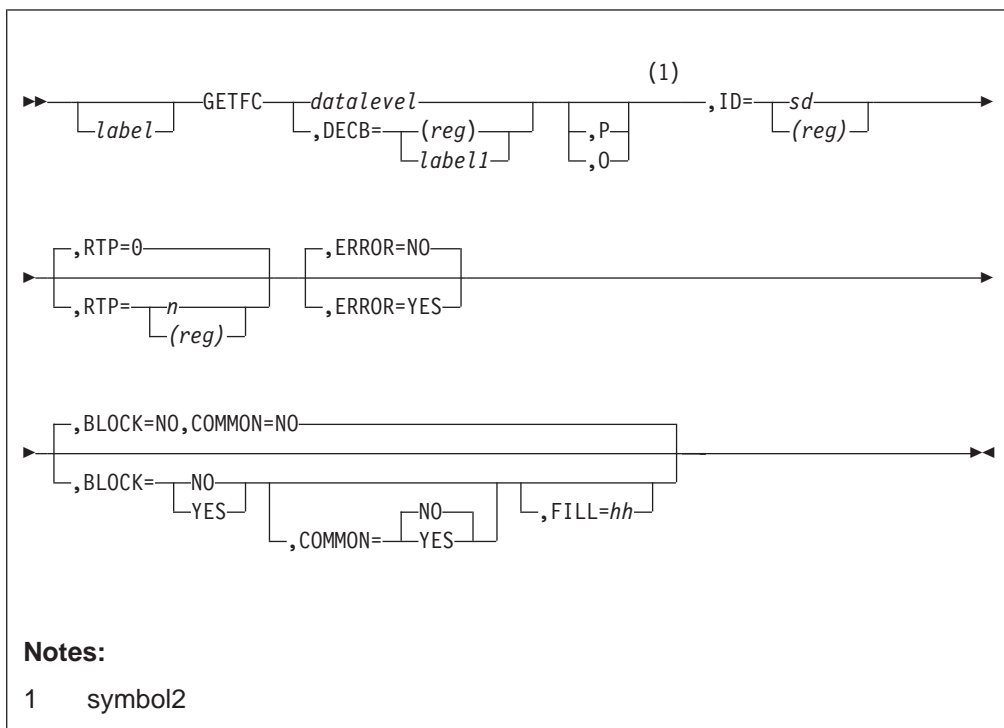
## Examples

- This invocation gets a block on data level 6. The size in bytes is contained in general register R4. No initialization is performed.  
GETCC D6,SIZE=(R4)
- This invocation gets a 1055-byte block on data level 3 from the shared storage area. No initialization is performed.  
GETCC D3,L2,COMMON=YES
- This invocation gets a 4KB block on data level 1 from the shared storage area. The block is initialized with X'00'.  
GETCC D1,L4,FILL=00,COMMON=YES
- This invocation gets a 381-byte block on data level 0 from the private storage area. The block is initialized with X'00'.  
GETCC D0,L1,FILL=00
- This invocation gets a block on data level E from the private storage area. The size of the block is calculated as the sum of two self-defining terms. The block is not initialized.  
GETCC DE,SIZE=LCP3880+LSSC
- This invocation gets a block on data level 5 from the private storage area. The block type is specified in register R0. The block is not initialized.  
GETCC D5,(R0)
- This invocation gets a 4KB block on data level 1 from the shared storage area. The block is initialized with X'00' and is given a protected storage key.  
GETCC D1,L4,FILL=00,COMMON=PROTECTED
- This invocation gets a block in the DECB specified by the address in register R7 from the private storage area. The block type is specified with a record ID of C'AA' and attribute 0 from the RIAT.  
GETCC DECB=(R7),ID=C'AA',RTP=0

## GETFC–Get File Pool Address and Storage Block

This general macro obtains an available file pool address and optional storage block based on the attributes of the specified record ID. The optional storage block may be accessed by other entry control blocks (ECBs) through common storage or it may be protected from common access.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *datalevel*

An ECB data level (D0–DF).

#### *symbol2*

If P is specified, RIAT attribute type 0 for the designated record ID applies. If O is specified, RIAT attribute type 1 for the record ID applies. See the RIATA information in *TPF System Generation* for further information on RTPx.

This parameter is optional and has been kept for migration purposes. New code using this macro should use the RTP parameter instead.

#### **DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB).

#### **RTP=n|(reg)**

A digit from 0–9 can be specified to designate the desired combination of record attributes assigned for the record specified in the ID parameter. RTP may also be a register containing the RIAT attribute (0–9). Valid registers are R0–R7. If a register is specified it should not be the same as the register specified for the ID or DECB parameters.



For more information about record attributes, see RIAT and RIATA in *TPF System Generation*.

**Note:** symbol2 and the RTP parameter are mutually exclusive.

### **BLOCK**

Specify one of the following:

#### **YES**

A storage block will be obtained on the designated level.

#### **NO**

No storage block is obtained.

The default is NO for this optional parameter.

### **ERROR**

This is an optional parameter.

#### **YES**

Return is made to the caller on an error.

#### **NO**

A system error with exit occurs on an error.

The default value is NO.

### **ID=sc[(reg)**

A required, self-defining term that represents a valid RIAT table record ID or a register containing a right-justified hexadecimal record ID. Valid registers are R0–R7. If a register is specified it should not be the same as the register specified for the RTP or DECB parameters.

### **COMMON**

Allocated block storage (BLOCK parameter) can be made accessible to all other ECBs or be protected from them by using this parameter.

#### **YES**

The storage comes from the pool of shared main storage. Any ECB can access the storage block.

#### **NO**

The storage is unique to this ECB, and is protected from reference or alteration by other ECBs.

NO is the default.

The COMMON parameter is used in combination with the BLOCK parameter. Authorization to use the COMMON must be established for the calling program (OPTIONS=(CMB)) in IBMPAL.

### **FILL=hh**

This is an optional parameter used in combination with the BLOCK parameter. If specified, a 1-byte hexadecimal value (hh) is supplied that will be used to initialize the user section of the storage block to be allocated.

For example, by coding FILL=00, a storage block allocated by the BLOCK parameter will be initialized to X'00's.

If the FILL parameter is not specified, the storage block contents will not be initialized and will not be predictable.

## GETFC

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The core block reference word (CBRW) on the specified ECB data level (*datalevel*) or DECB must not be holding a block if the optional BLOCK parameter is coded with YES.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown unless BLOCK=YES is coded; then R14 will contain the block address. The contents of all other registers are preserved across this macro call.
- If no error occurs, the record address assigned is placed in the specified file address reference word. The address and size of the block obtained is placed in the core block reference word specified, if BLOCK=YES. The condition code is unpredictable if ERROR=NO is coded. Otherwise, if ERROR=YES is coded, one of following conditions apply:
  - Condition Code 0, good return.
  - Condition Code 3, Record ID is invalid or the Record ID and RTP combination was not defined in RIAT.

### Programming Considerations

- This macro can be run on any I-stream.
- The input record ID must be defined in the Record ID Attribute Table.
- This macro cannot be used until after pools have been activated. So, for example, using this macro in 1052 state results in a system error.
- If both symbol2 and RTP parameters are omitted, the default setting for record attribute type is 0 (RTP=0).
- The CBRW on the specified ECB data level (*datalevel*) or DECB must not be holding a block if the optional BLOCK parameter is specified.
- Registers R0–R7 are restored upon return from the macro call. The contents of R14 and R15 are unpredictable unless BLOCK=YES is coded. Then, R14 will contain the block address.
- Storage reserved by specifying COMMON=YES is a scarce resource in the system. Its use should be restricted to situations when a storage block must be passed to other ECBs.
- Initializing a main storage block via the FILL option, or through any other method, takes more CPU resource than not initializing the block.
- This macro may result in the equivalent of a WAITC.
- File pool addresses that are acquired in a global transaction (that is, after a TXBGC macro call but before a TXCMC or TXRBC macro call) are released if the transaction is rolled back using a TXRBC macro call. However, if BLOCK=YES was specified, the storage block remains attached to the ECB.
- 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

- Either of the next macro calls gets a pool on data level 0 with record attributes of id 'OM' type RTP0 as defined in the RIAT table and a block, the address of which is returned in R14. If there is an error, control is returned to the calling segment

with condition code 3. The second invocation is in the older form being supported for migration purposes. The preferred version is the first invocation.

```
GETFC D0, ID=C'OM', BLOCK=YES, ERROR=YES, RTP=0
```

```
GETFC D0, P, ID=C'OM', BLOCK=YES, ERROR=YES
```

- This macro call gets a pool record on data level 0 with attributes of ID X'D6D4' found in register R5. If there is an error, a system error is issued.

```
GETFC D0, ID=X'D6D4', RTP=(R5)
```

- This macro call gets a pool record with attributes of record 1234 of type RTP0 on data level 0.

```
GETFC D0, ID=X'1234'
```

- This macro call has the record ID being passed in register R1 with the default record attribute type RTP0.

```
GETFC D0, ID=(R1)
```

- This macro call gets a pool record in the DECB specified by R7 with attributes of the record ID passed in register R1 of type RTP0.

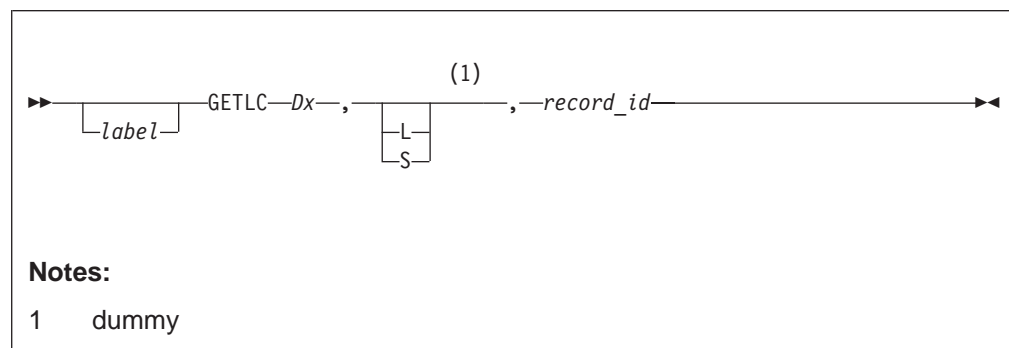
```
GETFC , DECB=(R7), ID=(R1)
```

## GETLC–Get Large File Storage Address

This general macro obtains an available file storage record address from a pool section of large records. The selection of a specific pool is based on the specified record ID. The address of the file storage record is stored into the file address reference word (FARW) of the specified data level.

**Note:** This macro is provided for backward compatibility only. New programs should be written using the GETFC macro (see “GETFC–Get File Pool Address and Storage Block” on page 248).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* An ECB data level (D0-DF) must be specified

*dummy*

This symbol is no longer required. However, it must be accounted for by inserting delimiting commas or a character such as an L or S as formerly required.

*record\_id*

A 2-character record ID must be specified as parameter 3. The assembly program uses the given ID to generate the V-constant of the macro expansion. The V-constant is resolved by the linkage editor program. The resolution of this V-constant determines whether a long term, short term, or duplicate pool address is returned.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The record ID used as *record\_id* must have been made available to the system allocator.

### 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.
- A record address is placed in the file address reference word of the specified data level.

## Programming Considerations

- This macro can be executed on any I-stream.
- Only one record address may be obtained at a time through this macro. Separate requests must be made for each record address.
- The record ID used as *record\_id* must have been made available to the system allocator.
- No record address may be released twice.
- All addresses retrieved via GETLC must ultimately be released via RELFC.
- This macro may result in the equivalent of an implied WAITC.
- File pool addresses that are acquired in a global transaction (that is, after a TXBGC macro call but before a TXCMC or TXRBC macro call) are released if the transaction is rolled back using a TXRBC macro call.

## Examples

```
GETLC D0,L,OM
```

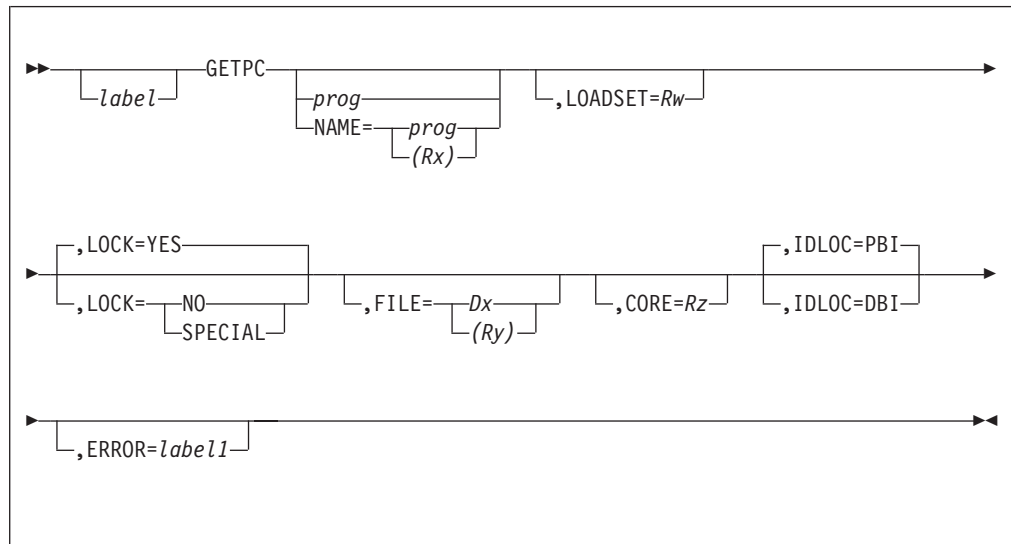
The address of a storage block of type OM is put into the file address reference word for data level D0. Whether the block is from the long term pool or the short term pool depends on the definition of the OM record returned. The L is merely a placeholder and has no other meaning.

## GETPC–Get Program and Lock in Storage

This general macro can be used to lock E-type programs in storage or to retrieve information on an E-type program. GETPC inspects the program allocation table (PAT) to determine if a program needs to be retrieved from file in order to lock it into storage. When the file resident program is no longer needed, it should be unlocked using the RELPC macro. A core resident program cannot be unlocked once a GETPC is issued.

A program's file or storage address can also be obtained from the PAT and returned.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*prog*

The name of the program that is to be locked in storage or located.

#### NAME

Besides using a positional parameter, the name of the program can also be provided using the NAME parameter.

**Note:** Either the prog parameter or the NAME parameter must be specified for E-type programs. These parameters are optional for C-type programs. If neither the prog or NAME parameter is specified for C-type programs, only a dsect for the GETPC parameter list is generated.

*prog*

The name of the program that is to be locked in storage or located.

*(Rx)*

A register (R0-R7) that contains the name of the program that is to be locked in storage or located.

#### LOADSET=Rw

A register (R1-R7) that contains the address of an 8-byte character string. This

character string is left justified, padded with blanks, and specifies the name of the loadset that contains the version of the program that is to be locked in storage or located.

If the character string contains all blank characters or if you do not specify this parameter, the version of the program associated with the requesting ECB is locked in storage or located. In this case, the name of the loadset that contains the version of the program associated with the requesting ECB is placed in the 8-byte character string.

**Note:** This option requires Restricted authorization.

## LOCK

An option that specifies whether or not the program will be locked in storage.

### YES

The requested program will be locked in storage. This is the default.

Programs allocated as PRIVATE can not have locks set.

### NO

The requested program will not be locked in storage. When LOCK=NO, FILE or CORE must be coded.

## SPECIAL

Specifies whether or not the program is to have the special lock indicator set.

Special locks set by GETPC must be unlocked using UNLOCK=SPECIAL in the RELPC macro. Programs allocated as PRIVATE can not have special locks set.

Programs locked through ZAPGM and ZRPGM have the special lock indicator set.

**Note:** This option requires Restricted authorization.

## FILE

An optional data level (D0-DF) whose File Address Reference Word (FARW) will receive the file address of the requested program.

When the FILE parameter is used, LOCK=NO should also be specified (to override the default of LOCK=YES). Without the LOCK=NO specification serious main storage depletion may result.

*Dx* Data level index (D0-DF).

*(Ry)*

A register (R0-R7) which contains the data level index. Can not be the same as coded for Rx above.

## CORE=Rz

An optional register (R0-R7) that will receive the storage address of the requested program.

## ERROR=label1

This optional label specifies a location where control is transferred if the program name being requested either cannot be found, was allocated as PRIVATE or already had a special lock indicator set. Register R15 will contain an error code indicating the type of error.

## GETPC

### IDLOC

An option that specifies whether the request will be serviced using the program base ID or the data base ID.

### PBI

The request will be serviced using the program base ID located in the entry control block field CE1PBI. This is the default.

### DBI

The request will be serviced using the data base ID located in the entry control block field CE1DBI. IDLOC=DBI is not valid when coded with LOCK=YES.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The register specified by the CORE keyword will contain the storage address of the requested program, with the following exception. When LOCK=NO, the register specified by the CORE keyword will return with 0 if the program is not already in storage, or the program is in storage and allocated as Private.
- The content of R14 and R15, except on an error condition when ERROR is coded, is unknown. The remaining operational registers are saved during the execution of the GETPC macro.
- If FILE was coded, then the specified FARW will be initialized with the program records generic ID and file address.
- If ERROR was coded and an error occurs, the indicated error routine is executed. Also, register R15 will contain an error indicator code. The following tags are generated by the GETPC macro and should be used for interrogating this error code:

IG\_LS\_ERR - The specified version of the program was not found.

IG\_NF\_ERR - The requested program was not found.

IG\_PP\_ERR - The requested program was allocated as PRIVATE.

IG\_RT\_ERR - A program retrieval error has occurred.

IG\_SL\_ERR - The SPECIAL lock indicator is already set.

**Note:** For special lock errors, processing will still return the requested data to the user.

## Programming Considerations

- This macro can be executed on any I-stream.
- Only one program at a time may be locked through this macro. Separate requests must be made for each program required.
- GETPC may be called with parameters to get a program or without parameters to generate a dsect. GETPC with parameters is primarily meant for E-type programs. A GETPC call with parameters generates an SVC. Consequently, C-type programs implementing SVC service routines should not use GETPC to get a program. C-type programs executing in problem state in an EVM space, however, may use GETPC to get a program. When C-type programs use GETPC to get programs, the NAME parameter must be used to specify the program name. The dsect provides labels for the GETPC parameter list and is used only by C-type programs.
- This macro does not attach the program to any ECB.



- GETPC will bring either core resident or file resident programs into storage.
- Private programs may not be locked in storage. When the requested program is allocated as PRIVATE and either LOCK=YES or LOCK=SPECIAL, a system error will result if ERROR is not coded.
- If LOCK=YES
  - and the specified program is file resident, then the program remains in storage until the RELPC macro is issued.
  - and the specified program is core resident, then the program is in storage and will remain there until the next IPL.
- If LOCK=SPECIAL
  - and the program is file resident, it will be locked into a common block. If the program is already in storage other than a common block it will be moved to a common block. The special indicator will be set. The common block will remain in storage until a RELPC UNLOCK=SPECIAL macro is issued.
  - and the program is in VFA, the program is moved to a common block.
- The SPECIAL lock will only be set once. Additional SPECIAL lock requests will return the error indicator IG\_SL\_ERR.

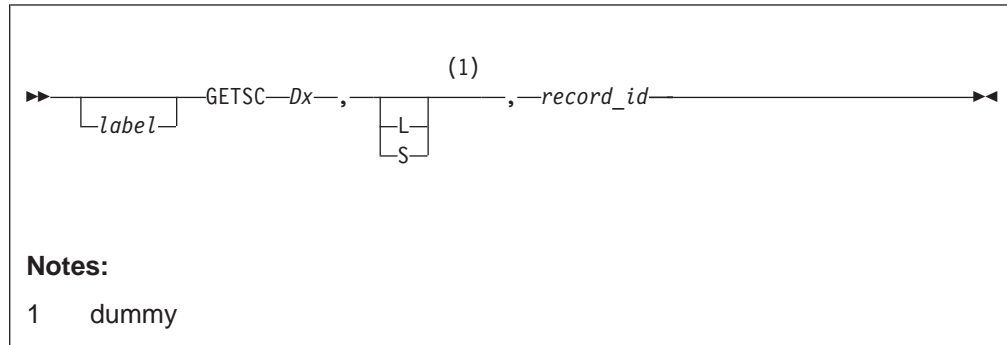
## Examples

None.

## GETSC–Get Small File Storage Address

This general macro obtains an available file storage record address from a pool section of small records. The selection of a specific pool is based on the specified record ID. The address of the file storage record is stored in the file address reference word (FARW) of the specified data level.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*Dx* Specify an ECB data level (D0-DF).

*dummy*

This parameter is no longer required. However, it must be accounted for by inserting the delimiting commas or a character such as L or S as formerly required.

*record\_id*

A 2-character record ID must be specified as parameter three. The assembly program will use the given ID to generate the V-constant of the macro expansion. The V-constant is resolved by the link editor program. The resolution of this V-constant determines whether a long, short, or duplicate address is dispensed.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The record ID used as *record\_id* must have been made available to the system allocator.

### 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.
- A record address is placed in the FARW of the specified data level.

### Programming Considerations

- This macro can be run on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB being processed.

- Only one record address may be obtained at a time through this macro. Separate requests must be made for each record address.
- All addresses obtained by a GETSC must ultimately be released by a RELFC macro.
- No record address may be released twice.
- The GETSC macro may result in the equivalent of an implied WAITC.
- The record ID used as record\_id must have been made available to the system allocator.
- File pool addresses that are acquired in a global transaction (that is, after a TXBGC macro call but before a TXCMC or TXRBC macro call) are released if the transaction is rolled back using a TXRBC macro call.

## Examples

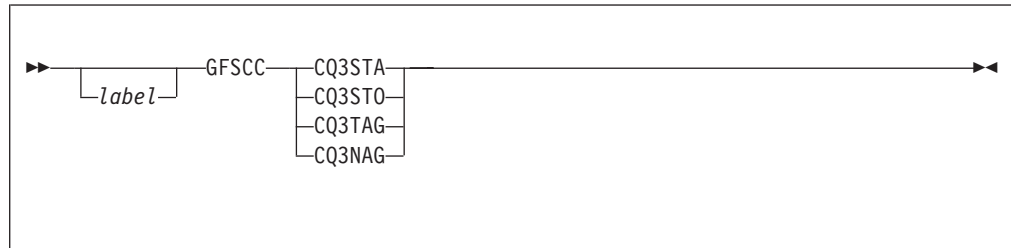
GETSC D0,L,OM

The address and size of a storage block of type OM is put into the file address reference word for data level D0. Whether the block is from the long term pool or the short term pool depends on the definition of the OM record returned. The L is merely a placeholder and has no other meaning.

## GFSCC–Initiate GFS Control

This general macro is available to file pool programs to run ancillary functions by the Get File Storage/Replace File Storage programs. The functions apply to file pool sections that exist in the system.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### **CQ3STA**

To start recording each file storage address dispensed for GETLC, GETSC or GETFC requests.

#### **CQ3STO**

To stop the GFS recording function.

#### **CQ3TAG**

To start tagging each long term file storage address returned with the RELFC macro. The 4-byte BCD name of the program using RELFC macro is used to tag the address.

#### **CQ3NAG**

To stop the returned file storage address tagging function.

### 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 executed on the main I-stream only.
- A CQ3STO request must be used to stop the GFS logging function. This will also initiate writing the last file storage activity file and the last dispensing activity log record (CY6TR) on a real-time tape.
- If a system restart occurs while parameter CQ3STA is in effect, GFS will reinstate the logging function.
- **This macro is restricted for use by file pool programs. The misuse of this macro could be detrimental to the system.**

### Examples

GFSCC CQ3STA

This invocation starts recording file addresses dispensed for GETLC GETSC or GETFC macro requests.

## GLMOD—Change Global Protect Key

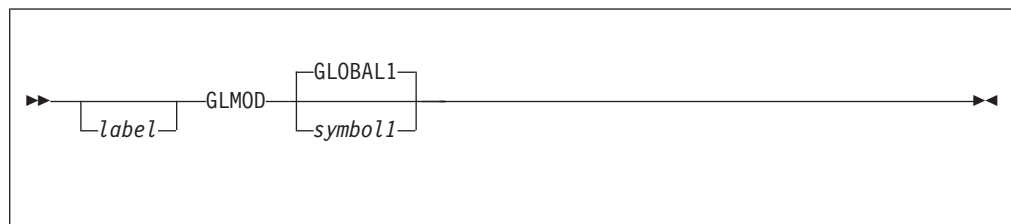
Use this macro to modify any global fields, core resident application data records, and tables, which reside in an area of main storage that carries a storage protection key different from that in the PSW used for application programs. This macro changes the storage protection key in the PSW for the application program concerned to match that of the global area and core resident data.

GLMOD performs this function by generating a KEYCC macro. To maintain uniformity, and flexibility in case of future system changes, application programs should use GLMOD, not KEYCC.

The PSW storage protection key must be restored after core modification by using the file keyword macro (FILKW). Between issuing GLMOD and FILKW, the application program must not issue any ENTER or WAITC macros (which may alter the PSW key), nor can it store data in working storage (ECB, core blocks).

See also the global area program material in the *TPF System Installation Support Reference*.

## Format



*label*

A symbolic name can be assigned to the macro statement.

*symbol1*|**GLOBAL1**

*symbol1* is a symbol equated to a storage protection key in CPSEQ. If omitted, GLOBAL1 is assumed.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction.
- The PSW storage protection key for this entry is changed to that equated to (*symbol1*) in CPSEQ.
- The condition code is unchanged by GLMOD.
- 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 executed on any I-stream.
- No changes can be made to any main storage area other than the global fields and core-resident application data records and tables.

## Examples

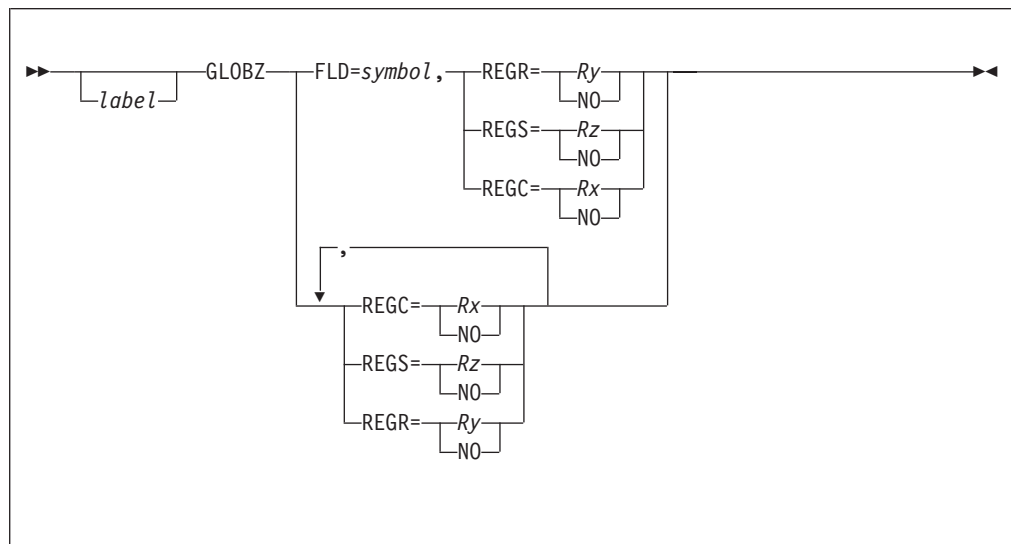
```
GLMOD
GLMOD GLOBAL3
```

## GLOBALZ—Define Global Fields

This general macro calls the global definition macros for global areas 1 or 3 (or both), and establishes addressability for the user by loading a specified base register if desired. The existing global fields for global area 1 are defined in the GLOBALA macro. The existing global fields for global area 3 are defined in the GLOBALY macro.

See the global area program material in the *TPF System Installation Support Reference*.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**FLD=symbol**

This is an optional keyword.

If coded, *global symbol* will be used to determine the global area in which the record descriptor is stored.

Any one of the register keywords must be specified. Those not specified default to REGx=NO.

**REGC=Rx|NO**

The register designated by Rx is loaded with the address of the GLOBALY area belonging to the first subsystem user of the subsystem. This keyword should be used for establishing addressability to SSU-common fields.

If coded with NO, only the GLOBAL3 DSECT macros are called; no register is loaded.

**REGR=Ry|NO**

If coded with Ry then segment GLOB defines all the DSECTs associated with the GLOBAL1 area; namely, GLOBALA, GLOBALB, GLOBALC, GLOBALD, GLOBE, GLOBALF, and GLOBALG. In addition, the register designated by Ry is loaded with the address of GLOBALA.

If coded with NO, only the GLOBAL1 DSECT macros are called; no register is loaded.



At least one of the register keywords REGx must be specified.

**REGS=Rz|NO**

This is an optional keyword.

If coded with Rz the GLOBAL3 DSECTs are defined and the register designated by Rz is loaded with the address of GL0BY.

If coded with NO, only the GLOBAL3 DSECT macros are called; no register is loaded.

At least one of the register keywords REGx must be specified.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- At least one REGx parameter must be present.

## Return Conditions

- Control is returned to the next sequential instruction.
- The registers defined as the operands of the REGx keywords are loaded with the addresses of the appropriate global areas.

The contents of all other registers are preserved across this macro call.

## Programming Considerations

- This macro can be executed on any I-stream.
- When the FLD parameter is specified, the address of the global area where the field resides is loaded into the register specified regardless of the register keyword supplied.

For example, if FLD=@GBFLD,REGS=R1 is coded and the @GBFLD field actually resides in global area 1, then R1 is loaded with the base address of global area 1, even though REGS (indicating global area 3) was specified.

## Examples

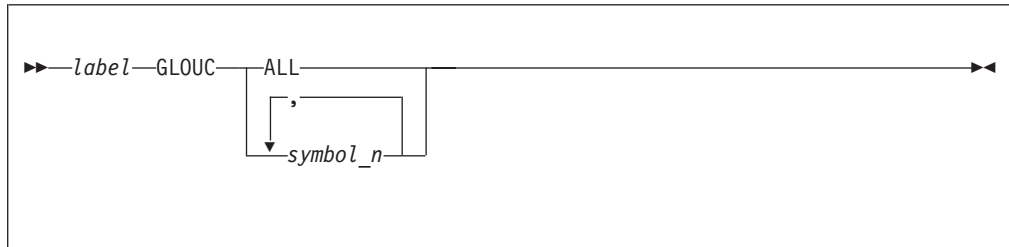
```
GLOBZ FLD=@NSCKAA,REGR=R1
```

R1 will be loaded with the address of the global area in which @NSCKAA resides.

## GLOUC—Request Keypoint Filing

This general macro supersedes the KEYUC and CINFC macros for requesting the filing of primary keypoint records and of keypointable application records whose memory and file pointers are in the Primary Global Directory (GL0BA). In addition, it provides the facility for requesting the filing of keypointable application records and special program records whose memory and file pointers are in the Overflow Global Directory (GL0BY).

### Format



*label*

A symbolic name can be assigned to the macro statement.

**ALL**

All keypointable global fields, keypoint B, keypoint D, keypoint E, and keypoint 1 will be filed for all I-streams.

*symbol\_n*

From 1–8 records, keypointable application record or special program record as defined in GL0BA or GL0BY, or primary keypoint records.

Valid parameters for records in GL0BA can be found in KEYUC. Valid parameters for special records and records in GL0BY can be found in GLOUC. Valid parameters for keypoints can be found in CINFC (K option).

### Entry Requirements

- All entry requirements are fulfilled by the proper use of the parameters as defined above.
- R9 must contain the address of the ECB being processed.

### 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 global directory item for each record referenced by *symbol\_n* will have an indicator bit set to request filing.
- The condition code is not changed.

### Programming Considerations

- This macro can be executed on any I-stream.
- GLOUC may be used without parameters to request filing of any records whose pointers are in the Overflow Global Directory (GL0BY) and whose 'request filing' indicator has been set by some other means. The FILKW macro may be issued

to restore the application program protection key to working storage, in addition to requesting filing of specified records. When primary keypoints are being filed, the following keypoints are in Group 1:

CTKB, CTKE, CTKD, and CTK1

Group 2 contains no keypoints.

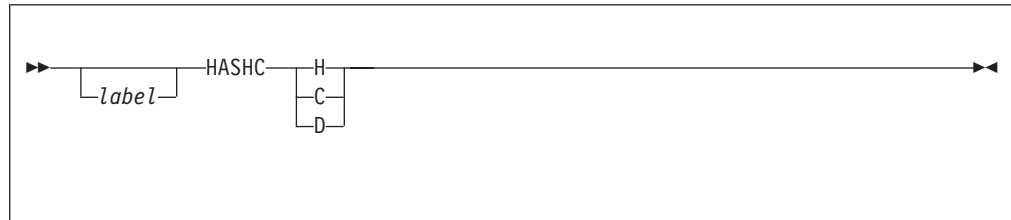
## Examples

```
GLOUC KEYB
GLOUC
GLOUC ALL
```

## HASHC—Calculate Record Slot Number

This general macro is used as an aid in calculating a slot index for any data base that requires, and is designed for, a compression algorithm. When given a unique key from a predefined maximum set of keys, this macro returns a number that can easily be manipulated to yield a slot index in a compressed record space. The macro is independent of any data base, but each data base must be designed for a compression algorithm.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**H** A hexadecimal number (4 bit digits).

**C** A character representation (0-9, A-Z).

**D** A decimal number (packed decimal, 4 bit digits).

### Entry Requirements

- R15 must contain the address of the data (unique input key) to be passed.
- R14 contains the count of the number of valid digits. The maximum number of digits that can be passed is 16 (for hexadecimal representation), 7 (for character representation), or 25 (for decimal representation).
- R9 must contain the address of the ECB being processed.

### Return Conditions

- Assuming normal return, R15 contains the “hashed” number which, when divided by the number of items allocated to the record space, yields the index for that record. If you specify the C parameter, the data pointed to by R15 on input is translated on return. On error return, R15 is zeroed.
- The contents of R14 are unknown. The contents of all other registers are preserved across this macro call.
- Control is returned to the next sequential instruction.

### Programming Considerations

- This macro can be executed on any I-stream.
- The data base must be designed to use a compression algorithm.
- This macro is available for use by any ECB-controlled program.

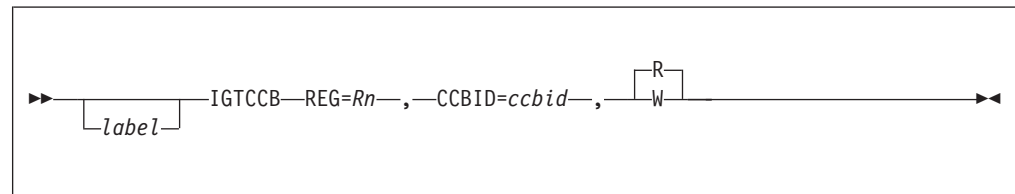
### Examples

None.

## IGTCCB—Get a Conversation Control Block Entry

This general macro returns a conversation control block (CCB) entry address in a specified register when given a CCB identifier.

### Format



*label*

A symbolic label may be assigned to this statement.

**REG=Rn**

The variable *Rn* specifies the register that is to contain the CCB entry address upon return. The valid range is R1-R7.

**CCBID=ccbid**

The variable *ccbid* specifies the address of the 4-byte field containing the CCB ID.

**R|W**

A positional parameter to indicate what type of operation is to be done on the CCB entry. This parameter corresponds to the read/write parameter on the CINFC Macro.

**R** Indicates read only capability is needed. This is the default.

**W** Indicates write/update capability is needed.

### Entry Requirements

The calling segment must set up addressability of the CCB.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of R0 - R7 are preserved across this macro call.
- The address of the CCB entry is returned in the specified register. When an error is encountered, the hexadecimal return value in the specified register is either zeros, indicating a bad CCB ID, or ones, indicating a bad TCB ID.

### Programming Considerations

- This macro can be executed on any I-stream.
- Since the CCBs can be allocated above the 16 megabyte line, the calling program must be in 31 bit addressing mode.

### Examples

- The first invocation identifies R1 as containing the CCB address for EBW000 upon return. Since the default operation is read access only, these two examples are the same.

## IGTCCB

```
IGTCCB REG=R1,CCBID=EBW000
IGTCCB REG=R1,CCBID=EBW000,R
```

- This invocation also identifies R1 as containing the CCB address for EBW000 upon return. In this case EBW000 is to be written or updated.

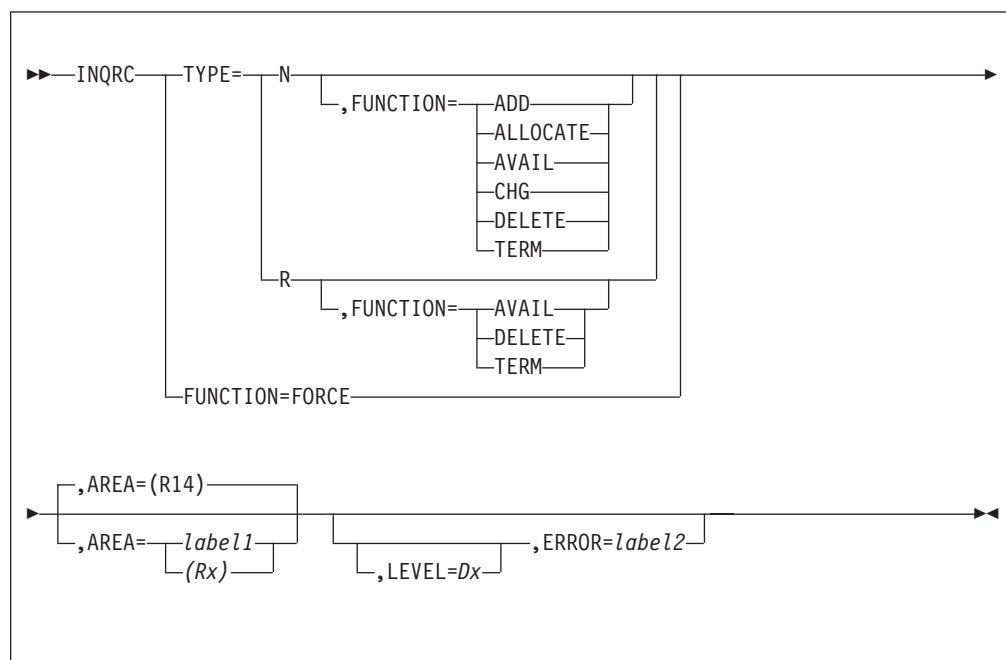
```
IGTCCB REG=R1,CCBID=EBW000,W
```

## INQRC—Convert Resource Application Interface

This general macro can be used to obtain the following:

- Resource identifier (RID) or session control block identifier (SCBID)
- Network qualified name (NQN)
- Resource definition bytes.

### Format



### TYPE

This indicates whether an RID (or SCBID) or a name is to be used as the search argument to locate the data.

**R** This indicates that an RID or SCBID is to be used as a search argument. The RID or SCBID is right-justified in a 4-byte field padded to the left with binary zeros. If the search argument is an SCBID, the corresponding SCB must be chained off an RVT entry.

**N** This indicates that a name is to be used as the search argument.

### AREA=label1(Rx)

This is a label or register (R0–R7, R14, or R15) containing the address of a work area that contains the input data and that is used to return the output if LEVEL is not coded. When it is used as a return area, 30 bytes must be reserved. If omitted, R14 is the default.

### LEVEL=Dx

This specifies the ECB level, D0 through DF, that is used to return the requested data. On return to the caller, the level contains a core block with the data requested. The core block contains a 16-byte standard header followed by the returned data.

### ERROR=label2

This is a label to branch to if the RID or SCBID or name provided is not valid or cannot be found. This parameter is required when LEVEL is coded.

**FUNCTION**

This is restricted to system use only.

**ADD**

Creates a resource definition for the new LU resource specified in the AREA input data.

**ALLOCATE**

Creates a resource definition for the new non-LU resource specified in the AREA input data.

**AVAIL**

Places the RVT entry for the LU resource specified in the AREA input data on the RVT available list.

**CHG**

Changes the name of the non-LU resource specified in the AREA input data to the new name specified in the AREA input data.

**DELETE**

Deletes the resource definition for the non-LU resource specified in the AREA input data.

**FORCE**

Removes all of the RVT entries from the RVT termination list and places them on the RVT available list.

**Note:** Do not specify the TYPE parameter if you specify the FORCE option.

**TERM**

Places the RVT entry for the LU resource specified in the AREA input data on the RVT termination list.

## Entry Requirements

- Format of the data specified for the AREA parameter when TYPE=R and FUNCTION=AVAIL|DELETE|TERM:

| Offset |     | Length | Description                                                                  |
|--------|-----|--------|------------------------------------------------------------------------------|
| Dec    | Hex |        |                                                                              |
| 0      | 0   | 1      | Reserved. The user must set this field to X'00' each time the macro is used. |
| 1      | 1   | 3      | Three-byte RID or SCBID                                                      |
| 1      | 1   | 1      | Reserved                                                                     |
| 2      | 2   | 2      | Two-byte RID                                                                 |

- Format of the data specified for the AREA parameter when TYPE=N and FUNCTION=ADD|ALLOCATE|AVAIL|DELETE|TERM:

| Offset |     | Length | Description                                                                                                              |
|--------|-----|--------|--------------------------------------------------------------------------------------------------------------------------|
| Dec    | Hex |        |                                                                                                                          |
| 0      | 0   | 8      | Network identification, left-justified and padded with blanks; when unqualified, this field must contain blanks (X'40'). |
| 8      | 8   | 8      | Name, left-justified and padded with blanks.                                                                             |

- Format of the data specified for the AREA parameter when TYPE=N and FUNCTION=CHG:



| Offset |     | Length | Description                                                                                                                           |
|--------|-----|--------|---------------------------------------------------------------------------------------------------------------------------------------|
| Dec    | Hex |        |                                                                                                                                       |
| 0      | 0   | 8      | Old network ID, left-justified and padded with blanks. If the resource name is not qualified, this field must contain blanks (X'40'). |
| 8      | 8   | 8      | Old resource name, left-justified and padded with blanks.                                                                             |
| 16     | 10  | 8      | New network ID, left-justified and padded with blanks. If the resource name is not qualified, this field must contain blanks (X'40'). |
| 24     | 18  | 8      | New resource name, left-justified and padded with blanks.                                                                             |

## Return Conditions

- Control is returned to the next sequential instruction.
- Register 14 contains the pointer to the returned data.
- Format of data returned by INQRC:

| Offset |     | Length | Label    | Description                                                          |
|--------|-----|--------|----------|----------------------------------------------------------------------|
| Dec    | Hex |        |          |                                                                      |
| 0      | 0   | 1      | INQDATA  | Reserved (always X'00')                                              |
| 1      | 1   | 3      | INQRID3  | Three-byte RID or SCBID                                              |
| 1      | 1   | 1      | Reserved |                                                                      |
| 2      | 2   | 2      | INQRID   | Two-byte RID                                                         |
| 4      | 4   | 8      | INQNETID | Network identifier                                                   |
| 12     | C   | 8      | INQNAME  | Resource name                                                        |
| 20     | 14  | 2      |          | Reserved for future use by IBM                                       |
| 22     | 16  | 1      | INQMODE1 | Mode byte 1 (see note 1)                                             |
| 23     | 17  | 1      | INQMODE2 | Mode byte 2 (see note 1)                                             |
| 24     | 18  | 1      | INQMODE3 | Mode byte 3 (see note 1)                                             |
| 25     | 19  | 1      |          | Reserved for future use by IBM                                       |
| 26     | 1A  | 1      | INQNTYP1 | Node type byte 1 (see note 1)                                        |
| 27     | 1B  | 1      | INQNTYP2 | Node type byte 2 (see note 2)                                        |
| 28     | 1C  | 1      | INQCPUID | CPU ID of resource                                                   |
| 29     | 1D  | 1      | INQRTNCD | Return code (see note 4).                                            |
|        |     |        | INQERROR | Resource was not found (see note 3) (X'04').                         |
|        |     |        | INQNORSC | No resources are available for the ADD or ALLOCATE function (X'40'). |
|        |     |        | INQDUPL  | Name unqualified, duplicates exist (X'80').                          |

### Note:

1. For details on possible values, see the NODEQ macro.
2. For details on possible values, see the TRMEQ macro.
3. If a valid SCBID is entered but the SCB is not chained to an RVT, INQERROR is returned.
4. There is no label for a return code of X'00'.

## INQRC

- The contents of scratch register R15 are unknown.
- The contents of the remaining operational registers are saved during the execution of this macro.

## Programming Considerations

- The ECB reference register (R9) must contain the address of the ECB being processed before using this macro.
- When unqualified names are used, there can be synonyms, since the name is guaranteed unique only within a network. If an unqualified name is used and there is more than one resource with the same name, the data associated with the first name located is returned. RVT entries whose NETIDs are blank are unique and ignore NETID checks. The data is returned only when the names match.
- Detection of duplicate names is not considered an error and does not activate the error branch.

## Examples

If the input is set up as follows:

- EBW000 = 'TPFNET '
- EBW008 = 'LU1 '

and you code: **INQRC TYPE=N,AREA=EBW000,LEVEL=D5,ERROR=ROUTINE**  
the output would be:

- EBW000 = 'TPFNET ' (unchanged)
- EBW008 = 'LU1 ' (unchanged)
- The core block on level 5 =

| Bytes | Contents |
|-------|----------|
|-------|----------|

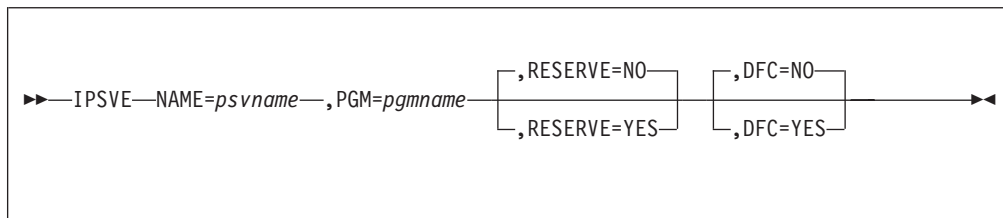
|       |                                     |
|-------|-------------------------------------|
| 0–15  | Standard header                     |
| 16    | Reserved (always X'00')             |
| 17–19 | Three-byte RID value for TPFNET.LU1 |
| 20–27 | Network identifier (TPFNET)         |
| 28–35 | Resource name (LU1)                 |
| 36–37 | Reserved for future use by IBM      |
| 38–40 | Mode bytes                          |
| 41    | Reserved for future use by IBM      |
| 42–43 | Node type bytes                     |
| 44    | CPU ID                              |
| 45    | Return code.                        |

## IPSVE—Create a PSV Table Entry

Use this general macro to create a process selection vector (PSV) table entry. Use the IPSVT macro to define the start and the end of the PSV table; see “IPSVT—Define the Start and End of the PSV Table” on page 277 for more information.

See *TPF ACF/SNA Network Generation* and *TPF ACF/SNA Data Communications Reference* for more information about process selection vectors (PSVs).

## Format



### **NAME=psvname**

specifies the PSV name, where *psvname* is a 1- to 6-character alphanumeric name. The PSV names that are reserved for IBM use start with the letter I.

### **PGM=pgmname**

specifies the program associated with the PSV name, where *pgmname* is the 4-character program name. This program can be file resident. It must reside in the basic subsystem (BSS), and it can also reside in other subsystems.

### **RESERVE**

specifies one of the following:

#### **YES**

specifies that the entry is reserved for future use. This parameter generates a SERRC macro call in place of the ENTNC macro call.

#### **NO**

specifies that the entry is not reserved for future use.

### **DFC**

specifies one of the following:

#### **YES**

specifies that the user PSV provides the SNA-designed data flow control (DFC) layer.

#### **NO**

specifies that the user PSV does not provide the SNA-designed data flow control (DFC) layer.

## Entry Requirements

Segment COBU, which generates the PSV table, must be allocated as residing in the basic subsystem (BSS).

## Return Conditions

None.

## IPSVE

### Programming Considerations

- In a loosely coupled system, the PSV table is shared by all processors in the complex.
- Although segment COBU can be loaded using the online loader, the new copy of this segment is not in effect until the next system initial program load (IPL) is performed.

### Examples

The following example defines a PSV table.

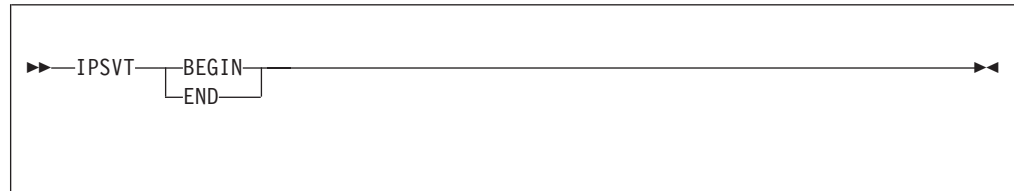
```
IPSVT BEGIN
IPSVE NAME=TYMNET,PGM=TYMI
IPSVE NAME=SITA1,PGM=SITI
IPSVT END
```

## IPSVT—Define the Start and End of the PSV Table

Use this general macro to define the start and end of the process selection vector (PSV) table. Use the IPSVE macro to create an entry in the PSV table; see “IPSVE—Create a PSV Table Entry” on page 275 for more information.

See *TPF ACF/SNA Network Generation* and *TPF ACF/SNA Data Communications Reference* for more information about process selection vectors (PSVs).

### Format



#### BEGIN

produces the PSVTBL label, which indicates the start of the PSV table.

#### END

produces the byte constant that represents the number of IPSVE entries generated.

### Entry Requirements

Segment COBU, which generates the PSV table, must be allocated as residing in the basic subsystem (BSS).

### Return Conditions

None.

### Programming Considerations

- In a loosely coupled system, the PSV table is shared by all processors in the complex.
- Although segment COBU can be loaded using the online loader, the new copy of this segment is not in effect until the next system initial program load (IPL) is performed.

### Examples

The following example defines a PSV table.

```
IPSVT BEGIN
IPSVE NAME=TYMNET,PGM=TYMI
IPSVE NAME=SITA1,PGM=SITI
IPSVT END
```

## ITPNT–Transaction Program Name Table Macro

Use this general macro to:

- Define entries in the transaction program name table (TPNT).

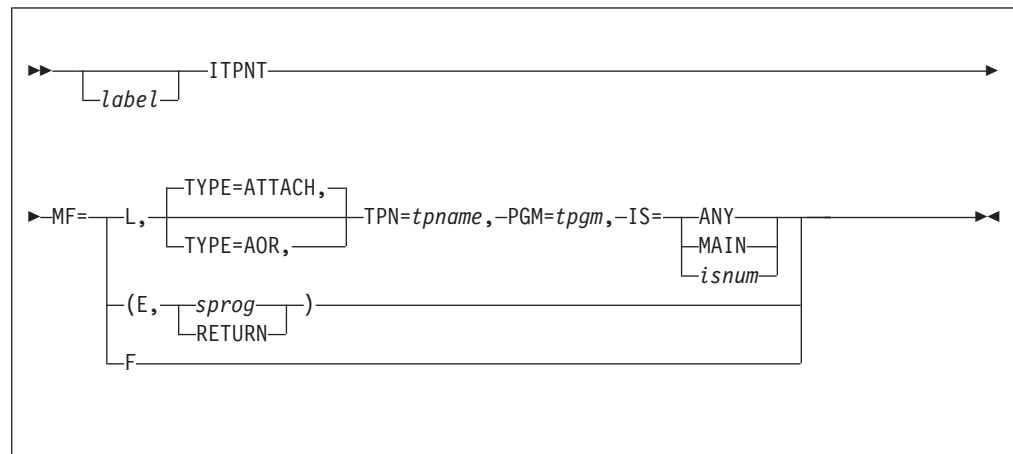
This is one of the initial tasks you must perform to use the TPF/APPC support package. The TPNT defines the local TPF transaction program names that are activated by either the remote transaction programs, the TPF/APPC `ACTIVATE_ON_CONFIRMATION` verb, or the TPF/APPC `ACTIVATE_ON_RECEIPT` verb. The table resides in a user-replaceable program segment (CHQ0) and contains:

1. The transaction program name carried in the ATTACH message received from a remote LU 6.2 node, or issued on the `ACTIVATE_ON_CONFIRMATION` or `ACTIVATE_ON_RECEIPT` verb.
2. The associated TPF E-type program segment that provides the application function.

See *TPF ACF/SNA Data Communications Reference* for more information about the tasks for installing TPF/APPC support.

- Generate the instructions to search the TPNT for a specified entry.

### Format



*label*

is a label for the ITPNT macro. The label is not used and is ignored if it is specified.

**MF**

specifies the macro form, which can be one of the following:

- L** generates a TPNT entry.
- E** generates the instructions necessary to search the TPNT for a specified TPF transaction name. This is the executable form of the ITPNT macro.
- F** generates the final TPNT entry for this segment.

**TYPE**

specifies the type of transaction program name being defined. The type determines how the transaction program can be activated. Specify one of the following:

**ATTACH**

When an ATTACH FMH5 containing the transaction program name specified on the TPN parameter is received from a remote LU, TPF/APPC support uses the SWISC macro to activate the program specified on the PGM parameter. The program that is activated must be designed to accept the TPF ATTACH interface.

**AOR**

When a transaction program issues the TPF/APPC ACTIVATE\_ON\_RECEIPT or ACTIVATE\_ON\_CONFIRMATION verb, TPF/APPC support uses the SWISC macro to activate the TPF real-time entry control block (ECB) segment specified on the PGM parameter.

If you specify TYPE=AOR, the values specified for both the PGM and TPN parameters must be identical and must name an allocated TPF E-type program name.

**Note:** See the “TPPCC–TPF/APPC Conversation Verb Macro” on page 414 for more information about the ACTIVATE\_ON\_RECEIPT and ACTIVATE\_ON\_CONFIRMATION verbs.

**TPN=***tpname*

specifies the local transaction program name known to the TPF system.

**Notes:**

1. If you specify TYPE=ATTACH, this entry defines the transaction program name carried in an ATTACH FMH5 record received from a remote transaction program. In this case, the length of *tpname* can be from 1 to 64 characters.
2. If you specify TYPE=AOR, this entry defines a transaction program segment name used with the TPF/APPC ACTIVATE\_ON\_RECEIPT and ACTIVATE\_ON\_CONFIRMATION verbs. In this case, *tpname* must be identical to the value specified for the PGM parameter.
3. If you specify TYPE=AOR and do not specify a value for this parameter, *tpname* defaults to the value specified for the PGM parameter.

**PGM=***tpgm*

specifies the name of the TPF E-type program associated with the transaction program name specified for the TPN parameter.

**Notes:**

1. If you specify TYPE=ATTACH, this is the name of the TPF program that is activated when an ATTACH FMH5 is received from a remote LU.
2. If you specify TYPE=AOR, this is the name of the TPF program that is activated when a message or other information is available to satisfy the TPF/APPC ACTIVATE\_ON\_RECEIPT and ACTIVATE\_ON\_CONFIRMATION verbs.
3. If you specify TYPE=AOR and do not specify a value for this parameter, *tpgm* defaults to the value specified for the TPN parameter.

**IS** specifies the I-stream on which the activate transaction program is executed. Specify one of the following:

**ANY**

specifies that the transaction program can run on any available I-stream. A SWISC macro is generated and the TPF system chooses any of the available I-streams. The SWISC load balancing decision is made at

## ITPNT

execution time and, therefore, a transaction program defined as IS=ANY can cause the defined transaction program to run on multiple I-streams concurrently.

### MAIN

specifies that the transaction program can run on only the main I-stream. Specify this parameter when the TPF system is generated to run in a single I-stream environment.

### *isnum*

specifies that the transaction program can run only on a particular I-stream, where *isnum* is the I-stream.

### *sprog*

is the TPF E-type program to be activated next to continue the search of TPNT entries if the transaction program name is not found in this part of the TPNT.

### RETURN

specifies that this is the last segment of the TPNT to be searched. If the requested transaction program is not found, control is returned to the caller with a *not found* indication.

## Entry Requirements

- The TPF/APPC support code uses the programs containing these tables to translate the transaction program name to a TPF real-time program segment.
- The program segments containing the ITPNT macros are activated to:
  - VERIFY a transaction program name, or
  - ACTIVATE the program segment associated with a transaction program name.

The VERIFY option is specified with a value of zero in R0 while the ACTIVATE option is specified with a value of four in R0.

- The following considerations apply when the program containing the ITPNT entries is activated:
  - R9 must contain the address of the entry control block (ECB) being processed.
  - For ATTACH processing, data level 0 (D0) contains the ATTACH FMH5 record.
  - For AOR processing, if the data level was specified on the TPF/APPC ACTIVATE\_ON\_RECEIPT or ACTIVATE\_ON\_CONFIRMATION verb, that data level is attached to the same data level on the activated ECB.
  - Data level 15 contains the TPF/APPC work block (IWBL).
  - ECB work area field EBW000 contains the name of the local TPF transaction program to be found. The field must begin with a 1-byte length of the transaction program name followed by the transaction program name.

## Return Conditions

- When the TPF programs containing these tables are activated, control is returned to the calling program only if the name of the transaction program cannot be found, or the VERIFY option was specified.
- If the transaction program name is found and the ACTIVATE option was specified, a SWISC macro is called to activate the associated TPF program. For ATTACH processing, data levels D0 and DF are released before the SWISC macro is called. For AOR processing, data level DF is released before the SWISC macro is called, and the data level specified on the original TPF/APPC



ACTIVATE\_ON\_RECEIPT or ACTIVATE\_ON\_CONFIRMATION verb is placed on the same data level for the newly activated ECB, or the TOKEN specified on the original verb is placed in EBX000.

- If control is returned to the calling program, a value of 0 is placed in R0 to indicate that the transaction program name was found, or a value of 4 is placed in R0 to indicate that the transaction program name was not found.
- See “Programming Considerations” for additional information on return conditions.

## Programming Considerations

- The ITPNT macro is used only in the TPF segments that are used to define the transaction program name table (TPNT).
- In the segments that are used to define the transaction program name table, the first ITPNT macro must specify the E option for the MF parameter to generate the code to search the TPNT. The local TPF transaction program names are then defined with the L option for the MF parameter. To indicate the last entry for this segment, the last ITPNT macro must specify the F option for the MF parameter.
- In a loosely coupled complex, if any remote LUs require sessions with only 1 TPF processor (single image view) and the TPF system will initiate conversations with these remote LUs, define the service transaction program as follows:  
 ITPNT TYPE=ATTACH,PGM=CHRP,TPN=TPF\_SERVICE\_TP  
 ITPNT TYPE=AOR,PGM=CHRM,TPN=CHRM

- If all of the local TPF transaction program names do not fit in the TPF segment CHQ0, additional program names can be added to a continuation segment. The TPF program name of the continuation segment is specified with the E option of the MF parameter. See the examples that follow.

**Note:** The first segment containing the TPNT entries must be CHQ0.

- If the requested TPF transaction program name is found, and the ACTIVATE option was specified, the SWISC macro is called to activate the TPF program. If the VERIFY option was specified, a value of 0 is placed in R0 before returning to the calling program.
- If the requested TPF transaction program name is not found and MF=(E,*sprog*) is specified, control is transferred to the program specified by *sprog* to continue the search.
- If the requested TPF transaction program name is not found and MF=(E,RETURN) is specified, control is returned to the caller with a value of 4 in R0.

## Examples

The following example shows a TPNT that is split across 2 real-time segments, CHQ0 and CHQ1. The first ITPNT macro call in CHQ0 specifies that the table is continued in segment CHQ1; if the TPF transaction name is not found in the entries defined in CHQ0, the application will continue looking in segment CHQ1. The first ITPNT macro call in CHQ1 specifies that this is the end of the table.

```

 BEGIN NAME=CHQ0
*
* Generate the table lookup code for this segment
*
 ITPNT MF=(E,CHQ1) Continue table in CHQ1
*
* Generate the TPNT entries for this segment
*
```

## ITPNT

```

 ITPNT MF=L, X
 TPN=FAREQUOTE, X
 PGM=FQFQ, X
 IS=ANY
*
* Define Program to handle ACTIVATE_ON_RECEIPT
*
 ITPNT MF=L, X
 TYPE=AOR, X
 TPN=CONT, X
 PGM=CONT, X
 IS=ANY
 ITPNT MF=L, X
 TPN=BAGGAGE, X
 PGM=BAGS, X
 IS=ANY
 ITPNT MF=L, X
 TPN=E_MAIL, X
 PGM=MAIL, X
 IS=ANY
 ITPNT MF=L, X
 TPN=FILETRANSFER, X
 PGM=XFER, X
 IS=ANY
 ITPNT MF=F
 LTORG
 FINIS
 END

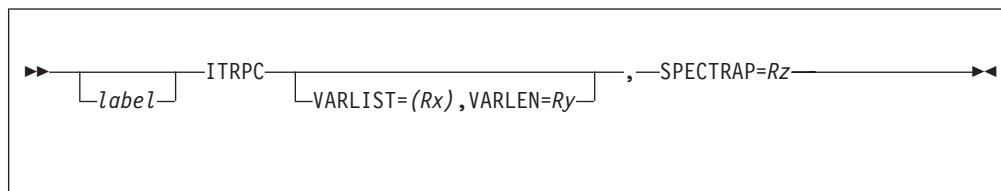
 BEGIN NAME=CHQ1
*
* Generate the table lookup code for this segment
*
 ITPNT MF=(E,RETURN) Last segment of TPNT
*
* Generate the TPNT entries for this segment
*
 ITPNT MF=L, X
 TPN=HOTEL_RESERVATIONS, X
 PGM=HOTL, X
 IS=MAIN
 ITPNT MF=L, X
 TPN=TEST_ECHO_PROGRAM, X
 PGM=TEST, X
 IS=ANY
 ITPNT MF=L, X
 TPN=TC_ISTREAM3_TEST, X
 PGM=TC3, X
 IS=3
 ITPNT MF=L, X
 TPN=REPORT_TRANSFER, X
 PGM=RPT0, X
 IS=ANY
 ITPNT MF=F
 LTORG
 FINIS
 END

```

## ITRPC—Send Simple Network Management Protocol User Trap

This general macro sends a Simple Network Management Protocol (SNMP) enterprise-specific trap to the remote SNMP destination managers specified in the `/etc/snmp.cfg` SNMP configuration file.

### Format



*label*

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

#### VARLIST

specifies a pointer to the variable binding list associated with the SNMP enterprise-specific trap being generated. See *TPF Transmission Control Protocol/Internet Protocol* for information about the variable binding lists and enterprise-specific traps used with SNMP agent support.

*Rx* is the register in the range R1–R7 that contains the pointer to the variable binding list.

#### VARLEN

specifies the length, in bytes, of the list specified with the VARLIST parameter.

*Ry* is the register in the range R1–R7 that contains the length of the variable binding list.

#### SPECTRAP

specifies an implementation-specific value for the SNMP enterprise-specific trap that describes the trap being generated. See *TPF Transmission Control Protocol/Internet Protocol* for information about the enterprise-specific traps used with SNMP agent support.

*Rz* is the register in the range R1–R7 that contains the value of the SNMP enterprise-specific trap.

### Entry Requirements

None.

### Return Conditions

If an attempt to build an SNMP enterprise-specific trap is successful, R0 will contain a return code of 0. If an attempt to build an SNMP enterprise-specific trap is unsuccessful, R0 will contain one of the following error returns:

Table 7. ITRPC Error Return

| Value Name              | Return Code | Description                                   |
|-------------------------|-------------|-----------------------------------------------|
| ISNMP_ITRPC_STATE_ERROR | –1          | The TPF system is not in CRAS state or above. |

## ITRPC

Table 7. ITRPC Error Return (continued)

| Value Name               | Return Code | Description                                                                                                                                                                                                                                                                                      |
|--------------------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISNMP_ITRPC_CONFIG_ERROR | -2          | The /etc/snmp.cfg SNMP configuration file was not refreshed successfully.<br><br>Do the following:<br>1. Create or fix the /etc/snmp.cfg SNMP configuration file.<br>2. Enter the ZSNMP command with the REFRESH parameter specified.<br>3. Issue the ITRPC macro again.                         |
| ISNMP_ITRPC_NO_TRAPS     | -3          | The /etc/snmp.cfg SNMP configuration file specifies that the TRAPIP keyword is coded as NONE, disabling SNMP trap support.<br><br>If you want to send traps, code the TRAPIP keyword on the /etc/snmp.cfg SNMP configuration file with a valid SNMP destination manager IP address or host name. |
| ISNMP_ITRPC_ENCODE_ERROR | -4          | An error occurred when encoding one of the trap values.                                                                                                                                                                                                                                          |
| ISNMP_ITRPC_SIZE_ERROR   | -5          | The size of the trap exceeded the maximum message size of 548 bytes.                                                                                                                                                                                                                             |

**Note:** These equates are defined in the ISNMP DSECT.

## Programming Considerations

- The TPF system must be in CRAS state or above to use this macro.
- The variable binding list must be in the correct SNMP basic encoding rules (BER) format. For information about BER encoding, see ISO 8825 *Part 1: Basic Encoding Rules*. Go to <http://www.iso.ch/> to view ISO 8825.
- Ensure that the /etc/snmp.cfg SNMP configuration file has been refreshed successfully by entering the ZSNMP command with the REFRESH parameter specified.
- Ensure that TCP/IP native stack support is defined to the TPF system. See *TPF Transmission Control Protocol/Internet Protocol* for information about defining TCP/IP native stack support.

## Examples

The following example sends enterprise-specific trap 5 as defined by the SPECTRAP parameter to all the SNMP destination managers specified in the /etc/snmp.cfg SNMP configuration file. This trap contains no variable binding list.

```
LA R1,5
ITRPC SPECTRAP=R1
```

## KARMA–SLC Link Alarm

This general macro is used to send messages to specified display terminals.

Messages can consist of:

- The user's own text
- A prepared message
- A prepared message completed by the user's substitution data.

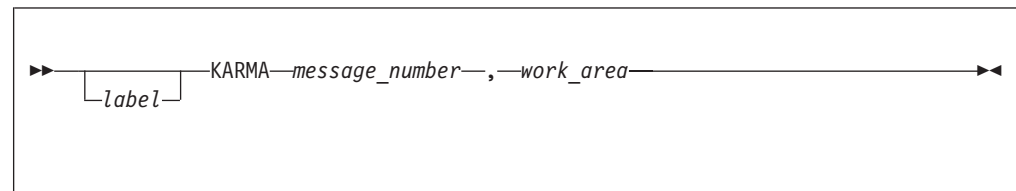
Messages may be preceded by a header containing any or all of the following:

- Time stamp
- Symbolic link number
- Symbolic link and channel number
- Symbolic line number.

Messages may be sent to any or all of the following addresses:

- Prime CRAS
- RO CRAS
- Address specified in EBROUT
- System Control Center
- Address specified by the user.

## Format



### *label*

A symbolic name can be assigned to the macro statement.

### *message\_number*

This specifies a prepared message number. Message numbers are contained in CMAMC. The X'00'–X'FE' specify "canned" messages. The CXJMOTX = X'FF' specifies its own text supplied.

### *work\_area*

This specifies the address of a 16-byte work area, aligned on a halfword boundary, which is immediately followed by text substitution data if required.

## Entry Requirements

- The work area is defined by the data macro CMAMC. Ensure the work area is aligned on a halfword boundary and set up the last four fields as follows before issuing a KARMA macro:

|        |                                                                                                                                                                                                                                            |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CMAOHD | If own text is specified or if the user wishes to override the canned header indicator this 1-byte field must contain the user's header indicator byte (See Programming Considerations). Otherwise, this byte must be set to zero.         |
| CMAOAD | If own text is specified or if the user wishes to override the canned addressee indicator, this 1-byte field must contain the user's addressee indicator byte. (See Programming Considerations). Otherwise, this byte must be set to zero. |
| CMAOCT | If own text is specified or if substitution data is required, this                                                                                                                                                                         |

## KARMA

halfword field must contain the character count of text or substitution data immediately following the work area.

### CMAOTX

If the addresses indicator byte specifies own address supplied, the first three bytes of this area contain the LNIATA of the terminal to which the message is to be sent.

If the message number specified was CXJMOTX—own text, this area will contain up to 80 bytes of text, including the EOM. If a canned message has been specified, this area may contain substitution data for completing the message.

- The following describes the ECB work area:

EBW000–EBWKAD

Fullword link keypoint address

EBW004–EBWCAD

Fullword channel keypoint address for channel oriented messages, that is those messages which require information pertaining to a particular line or channel, either in the message header, or as substitution data in the text of the message.

If the alarm/message is not channel oriented, the first byte of this field must contain X'FF'.

Level 6 must be available.

- Ensure register 1 (R1), R6, R7, R14, and R15 are available.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of registers R1, R6, R7, R14, and R15 are unpredictable. All other registers are unchanged.

## Programming Considerations

The KARMA macro enters CMC1, which sets up the work area and issues a CRETC macro (with the S parameter) to CMA1 to output the requested message.

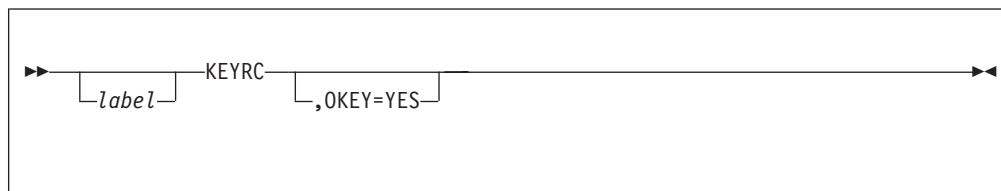
## Examples

None.

## KEYRC—Restore Protection Key

This general macro restores the current program status word (PSW) protection key to its normal value (protect key of working storage) or to the value saved by the CINFC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### OKEY=YES

Restores the original protection key that was saved by the previous CINFC macro call with the W option specified. See “CINFC—Control Program Interface” on page 66 for more information about the CINFC macro.

**Note:** The OKEY=YES parameter requires restricted authorization (OPTIONS=(RESTRICT)) on the IBMPAL macro.

### Entry Requirements

Register 9 must contain the address of the entry control block (ECB) being processed.

### Return Conditions

- On return, the current PSW protection key is identical to the key assigned to working storage (ECBs, data blocks, and fixed unprotected core).
- On return, when the OKEY=YES parameter is coded the current PSW protection key is identical to the PSW protection key before the previous CINFC macro call with the W option specified. See “CINFC—Control Program Interface” on page 66 for more information about the CINFC macro.
- 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 execution of this macro.

### Programming Considerations

- When this macro is used with the change protection KEYCC macro, it must be issued following the KEYCC macro and before:
  1. Storing into any of the normally unprotected areas (ECB, held data blocks, and fixed unprotected core storage)
  2. Issuing the WAITC macro or any macro with an implied wait
  3. Issuing a fast link macro.

See *TPF System Macros* for more information about the KEYCC macro. See “WAITC—Suspend Processing for ECB I/O Completion” on page 554 for more information about the WAITC macro.

## KEYRC

- When writing code to support application code that interfaces with system code or provides new functions, you must preserve the application environment. One environmental consideration is the storage protection key. It may not be correct to assume that the caller is always executing with the working storage protection key. The OKEY function is useful in restoring the protection key value that was in use by the application. You must use this function (by coding OKEY=YES) after the CINFC macro with the W option specified has been issued, but before invoking any system functions or programs that may, in turn, call the CINFC macro with the W option specified. When this is not possible, the saved protection key, which is in CE2OKEY, must be saved elsewhere and restored just before issuing the KEYRC macro with OKEY=YES coded.

See “CINFC—Control Program Interface” on page 66 for more information about the CINFC macro.

- This macro can be executed on any I-stream.

## Examples

None.





## LEVTA

### Return Conditions

- All registers are preserved across this macro call.
- If only the INUSE parameter is used, an ECB data level or DECB holding a core block will lead to a branch to the location specified by INUSE. If no core block is held at that ECB data level or DECB, control will be returned to the next sequential instruction (NSI).
- If only the NOTUSED parameter is used and no core block is held at the tested ECB data level or DECB, control will be returned to the symbolic location specified by the NOTUSED parameter. If a core block is held at the tested ECB data level or DECB, control will be returned to the NSI.
- If both NOTUSED and INUSE are used, control will be returned to the symbolic location specified by the parameter that satisfies the condition of the ECB data level or DECB tested.

### Programming Considerations

- This macro can be run on any I-stream.
- When the ECB data levels or DECBs are to be tested with the intent of releasing the core blocks for those ECB data levels or DECBs that have core blocks held, use the CRUSA macro.

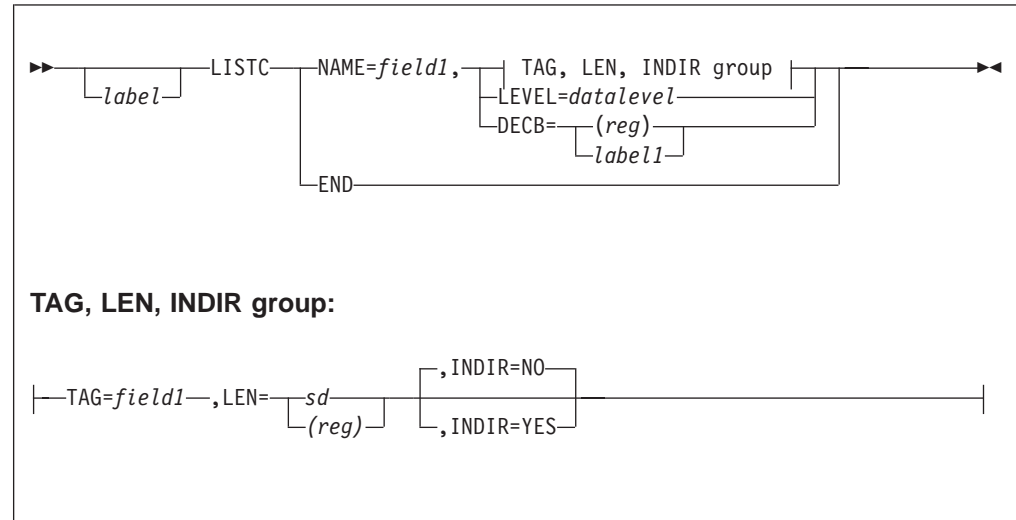
### Examples

None.

## LISTC–Dump Facility LIST Generator

This general macro is used to generate a list of data items for SNAPC and SERRC dump processing. The LISTC macro generates constants that identify the name, size, and location of the items that need to be included in the SNAPC/SERRC dump when the LIST parameter is supplied.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

The LISTC macro forces halfword alignment. This means that labels are coded as

```
label LISTC ...
```

or

```
label DS 0H
LISTC ...
```

Coding a label using an EQU statement instead of a DS statement can cause an error during processing of the SNAPC macro because of misalignment.

#### **TAG=field1**

This parameter specifies the location of the data to be dumped. This parameter is required unless the information is being supplied by the LEVEL parameter.

For example:

```
LISTC TAG=DATA
DATA DC C'THIS WILL BE DUMPED'
```

generates the list item to dump the field called DATA.

#### **NAME=field2**

This parameter specifies the name of the field to appear in the dump. This parameter is required, unless the information is being supplied by the LEVEL parameter. Specify a 1- to 8- byte character string. The name assigned may be, but is not required to be, the same name as the name of the label used in the TAG parameter. For example:

```
LISTC NAME=ITEM1,TAG=DATA
DATA DC C'THIS WILL BE DUMPED'
```

## LISTC

assigns the name ITEM1 to the data being dumped.

### LEN=*sd*(*reg*)

This optional parameter specifies the length of the field to be dumped. If omitted, it defaults to the length of the field specified in the TAG parameter. Specify a decimal number, any other self-defining term or expression or the general register containing the data length. For example:

```
LISTC NAME=ITEM1,TAG=DATA,LEN=L'DATA+4
DATA DC C'THIS WILL BE DUMPED'
 DC C'1234'
```

dumps the DATA field + four bytes.

### INDIR

This optional parameter specifies whether indirection is used to identify the location of the data to be dumped.

### YES

The TAG parameter specifies the location that contains the 4-byte address of the data to be dumped.

### NO

The TAG parameter specifies the location of the data to be dumped. ***This is the default.***

### LEVEL=*datalevel*

This parameter can be used by E-type programs in lieu of the TAG, LEN and INDIR parameters to identify a data area to be dumped. It identifies the data level referenced using R9, where the storage address (CE1CRx) and length (CE1CCx) of the data to be dumped can be found. Valid values for *datalevel* are D0, D1, ..., DE, DF. For SNAPC dumps, if the data level is not holding a block, then the LISTC macro call will be ignored.

### DECB=(*reg*)|*label1*

This parameter can be used by E-type programs instead of the TAG, LEN, and INDIR parameters to identify a data area to be dumped. It identifies the label or general register (R1–R7, R14 or R15) containing the address of the data event control block (DECB), where the storage address (IDECDAD) and length (IDECDLH) of the data to be dumped can be found. For SNAPC and SERRC dumps, the LISTC macro call will be ignored if the DECB address is found to be not valid or if the DECB is not holding a block.

### END

This parameter is mutually exclusive with all other parameters, and marks the end of the list of items specified by the LIST option of either SNAPC or SERRC. All LISTC lists must end with the END option.

For example,

```
LISTC END
```

ends a dump list.

## Entry Requirements

- If LEN=(Rx) is coded, Rx must contain the length of the data area to be dumped.
- If LEVEL=Dx is coded, R9 must reference the ECB for the data to be dumped, CE1CRx must contain the data address and CE1CCx must contain the data length.

- If the DECB=(reg) parameter is coded, R9 must reference the ECB for the data to be dumped, IDECDAD must contain the data address, and IDECDLH must contain the data length.

## **Return Conditions**

None.

## **Programming Considerations**

- The LISTC macro should be in the data constants section of the program. It should never be executed; it generates data — not executable code.
- If the location specified by the TAG parameter is in memory belonging to another ECB, even in VEQR mode, the dump contains the phrase Address Error for the LISTC with the erroneous address.

## **Examples**

None.

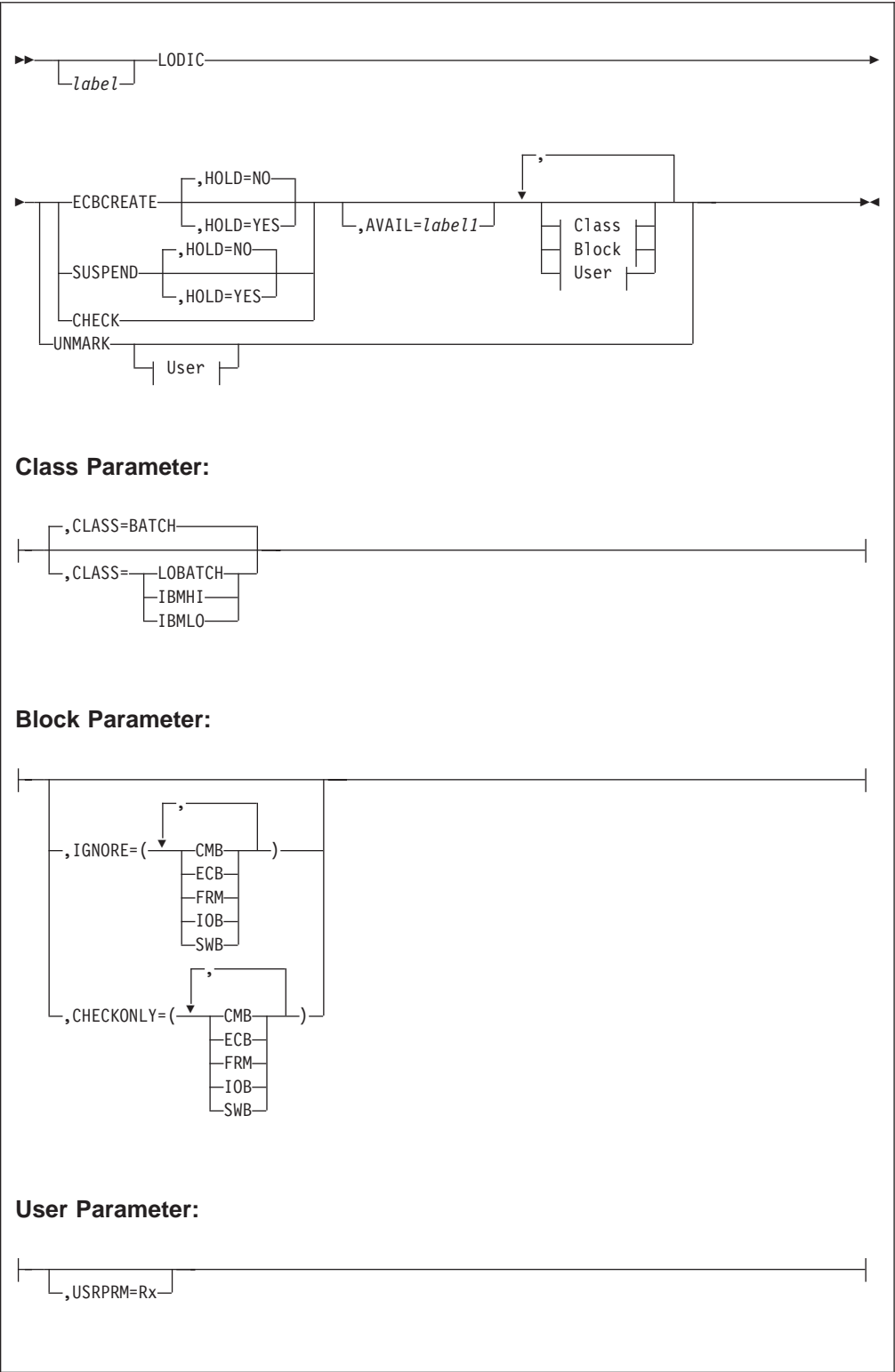
---

**LODIC–Check System Load and Mark ECB**

This general macro is used to check if enough system resources are available to begin processing low-priority or batch work, and to determine if an entry control block (ECB) can be suspended (based on the level of available resources).

An ECB that calls the LODIC macro with the ECBCREATE or SUSPEND parameters is marked as a low-priority ECB. Once marked, the ECB can be suspended when system resources are below the shutdown levels defined for a specified priority class (see Table 8). Even though the ECB is marked as being able to be suspended, the ECB cannot be suspended until it gives up control. In most cases the SUSPEND parameter forces the ECB to immediately give up control. The ECBCREATE parameter does not force the ECB to immediately give up control; the ECB must wait until it gives up control by entering a post-interrupt routine. Once suspended, the ECB does not receive control again until enough system resources are available.

Format



*label*  
A symbolic name may be assigned to the macro statement.

**CHECK**  
Checks are performed to see if the available system resources are above the

## LODIC

shutdown levels defined by the CLASS parameter (for example, checks to see if more work can be started). There is no immediate loss of control if you use the CHECK parameter. Control returns to the label specified by the AVAIL parameter (if coded) if more work can be started; otherwise control returns to the next sequential instruction. The CHECK parameter does not mark or unmark the ECB as capable of being suspended.

### ECBCREATE

The ECB is marked as capable of being suspended and a priority class is assigned to the ECB.

There is no immediate loss of control if the ECBCREATE parameter is used.

Checks are performed to see if the available system resources are above the shutdown levels defined by the CLASS parameter (for example, checks to see if more work can be started). Control returns to the label specified by the AVAIL parameter (if coded) if more work is allowed to be started; otherwise, control returns to the next sequential instruction.

**Note:** For compatibility with the TPF/MVS product, R14 will contain either 0 (no work can be started), or 1 (work can be started).

### SUSPEND

The ECB is marked as capable of being suspended and a priority class is assigned to the ECB.

The SUSPEND parameter causes the ECB to immediately lose control unless the ECB is holding a resource and HOLD=NO is coded.

Checks are performed to see if the available system resources are above the shutdown levels defined by the CLASS parameter (for example, checks to see if more work can be started). If the available system resources are below the shutdown levels and the ECB is able to lose control, then the ECB is immediately suspended. If the available system resources are above the shutdown levels, then the ECB is not immediately suspended. Control returns to the label specified by the AVAIL parameter (if coded) in all cases except one. The one case in which control does not return to the label specified by the AVAIL parameter is the case where the ECB does not lose control and the available system resources are below the shutdown levels. In this case, control always returns to the next sequential instruction.

### UNMARK

Removes the ability for the ECB to become suspended.

**Note:** An ECB marked as capable of being suspended remains marked as such until the LODIC macro with the UNMARK parameter is entered or until the ECB exits.

### AVAIL=*label*

The location to which to branch if the available system resources are above the shutdown levels defined by the CLASS parameter (for example, more work can be started). This keyword is used with the CHECK, ECBCREATE, and SUSPEND parameters. Without the AVAIL parameter, control always returns to the next sequential instruction.

### CLASS=BATCH|LOBATCH|IBMH|IBMLO

Assigns a priority class to the ECB. The priority class defines a set of unique shutdown values that are used to determine when the ECB will be suspended and when the ECB will be unsuspended. The classes are BATCH, LOBATCH,



IBMHI, and IBMLO. Priority classes BATCH and LOBATCH are supported for user programs. Priority classes IBMHI and IBMLO are reserved for E-type programs by the TPF system.

The CLASS parameter is used with the CHECK, ECBCREATE, and SUSPEND parameters.

**Note:** See Table 8 on page 299 for the shutdown levels of each priority class.

## **HOLD**

Indicates if the ECB can be suspended while holding a resource (FIWHC, CORHC, ENQC, TASNC, EVNWC, or SYNCC LOCK).

### **Notes:**

1. Does not apply to resources held with the \$LOCKC macro.
2. The HOLD=YES parameter requires restricted authorization (OPTIONS=(RESTRICT)) in the IBMPAL macro.

## **IGNORE**

Defines which block types are ignored when determining if the available system resources are above the shutdown levels defined by the CLASS parameter. The block types included will be ignored when determining if a running ECB should be suspended or if a suspended ECB should resume running. The IGNORE parameter is used with the CHECK, ECBCREATE, and SUSPEND parameters. Any subset of the following block types are supported:

### **CMB**

Common Block

### **ECB**

Entry Control Block

### **FRM**

4 KB Frame

### **IOB**

Input/Output Work Block

### **SWB**

System Work Block.

**Note:** The IGNORE keyword requires restricted authorization (CHECK=RESTRICT) in the IBMPAL macro.

## **CHECKONLY**

Defines which block types are checked when determining if the available system resources are above the shutdown levels defined by the CLASS parameter. Only the block types included are checked when determining if a running ECB should be suspended or if a suspended ECB should resume running. The CHECKONLY parameter is used with the CHECK, ECBCREATE, and SUSPEND parameters. Any subset of the following block types is supported:

### **CMB**

Common Block

### **ECB**

Entry Control Block

### **FRM**

4 KB Frame

## LODIC

### IOB

Input/Output Work Block

### SWB

System Work Block.

**Note:** The CHECKONLY keyword requires restricted authorization (CHECK=RESTRICT) in the allocator.

### USRPRM=*R*x

Allows user data to be passed from the application program to the LODIC service routine user exit. The specified register index is passed. R0 through R7 are valid.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The IGNORE and CHECKONLY keywords and the HOLD keyword specified with YES require the issuing program to have restricted macro usage authorization.

## Return Conditions

- For the ECBCREATE, SUSPEND, or CHECK parameters, control returns to the label specified by AVAIL, if AVAIL is coded and if resources are available for the specified priority class; otherwise, control returns to the next sequential instruction.
- For the UNMARK parameter, control always returns to the next sequential instruction.
- The return code depends on the shutdown levels defined for the specified priority class (see Table 8 on page 299).
  - R14 contains a value of 1 if resources are available for that priority class; otherwise, R14 contains a value of 0
  - R15 cannot be predicted.

## Programming Considerations

- This macro can be run on any I-stream.
- Once an ECB is marked as capable of being suspended, it becomes suspended whenever the ECB gives up control and the available system resources are below the defined shutdown levels (unless the ECB is holding a resource and HOLD=NO is coded). Once suspended, the ECB remains suspended until the available system resources return to levels above the defined shutdown levels.
- The create macros CREMC, CREDC, SWISC, CREXC, and CREEC will pass the LODIC parameters and the LODIC priority class information from the parent ECB to the child ECB. If the parent ECB is marked as capable of being suspended, the child ECB is marked in the same way. The other create macro, CRETC, does not pass this information.
- When you use a macro to create an ECB (such as CREMC or SWISC), the ECB is not created immediately. First a system work block (SWB) is placed on a list in the CPU loop. The ECB is not actually created until the SWB is dispatched from the list. Because of this delay, call the LODIC macro each time before creating a single ECB. The application issuing the LODIC macro must give up control before calling LODIC again to help ensure that ECBs created are given a chance to receive control and perform work. Otherwise, system resource depletion can result from too much work being scheduled and not started.
- ECBs marked with HOLD=YES may hold resources while suspended.

- Suspended ECBs that are using a large amount of resources can result in conditions in which these ECBs never get control back. Avoid having an ECB become suspended if it is using a large amount of resources.
- ECBs suspended by a LODIC macro call are purged during cycle-down to 1052 state unless they have been previously identified to survive cycle-down.
- Table 8 shows the shutdown levels of each priority class. The values given are the minimum percentage of each block type that must be available for ECBs to continue running.

In general, when the availability of a particular block type falls below a shutdown percentage defined in Table 8, ECBs labeled with the associated priority class will be suspended and will remain suspended until the availability of that block type rises above the shutdown percentage.

Table 8. Priority Class Table (Shutdown Levels)

| Priority Class                                                                                                                                                                  | Block Type |     |     |     |     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-----|-----|-----|-----|
|                                                                                                                                                                                 | CMB        | ECB | FRM | IOB | SWB |
| <b>LOBATCH</b>                                                                                                                                                                  | 96%        | 96% | 96% | 96% | 96% |
| <b>BATCH</b>                                                                                                                                                                    | 48%        | 48% | 48% | 48% | 48% |
| <b>IBMLO</b>                                                                                                                                                                    | 96%        | 96% | 96% | 96% | 96% |
| <b>IBMHI</b>                                                                                                                                                                    | 32%        | 32% | 32% | 32% | 32% |
| <b>Note:</b> <ul style="list-style-type: none"> <li>– These are the shutdown level values as shipped by IBM.</li> <li>– IBMLO and IBMHI are reserved for use by IBM.</li> </ul> |            |     |     |     |     |

## Examples

- The following example:
  - Forces the ECB to immediately give up control
  - Suspends the ECB whenever the number of available ECBs, SWBs, 4 KB frames, or CMBs falls below the shutdown level for the BATCH priority class
  - Does not allow the ECB to receive control until the number of available ECBs, SWBs, 4 KB frames, and CMBs are above that shutdown level
  - Allows the ECB to be suspended even if the ECB is holding a resource.

```
LODIC SUSPEND,CLASS=BATCH,IGNORE=(IOB),HOLD=YES,AVAIL=MOREWORK
```
- The following example:
  - Indicates that this ECB will be suspended whenever the number of available ECBs, 4 KB frames, or SWBs are below the shutdown level for the LOBATCH priority class
  - Returns control to the GOAHEAD label if more work can be started, (based on the number of available ECBs, SWBs, or FRMs); otherwise, returns control to the next sequential instruction.
  - Passes register 2 to the LODIC macro service routine user exit

```
LODIC ECBCREATE,CLASS=LOBATCH,CHECKONLY=(ECB,FRM,SWB),AVAIL=GOAHEAD,USRPRM=R2
```
- The following example:
  - Checks if there are enough system resources to start more work
  - Returns control to label GOSTART if available system resources are above the shutdown levels defined for the LOBATCH priority class; otherwise, returns control to the next sequential instruction.

```
LODIC CHECK,AVAIL=GOSTART,CLASS=LOBATCH
```

## LODIC

- The following example sets this ECB so it cannot be suspended again unless another LODIC or TMSLC macro is issued.

```
LODIC UNMARK
```

- As soon as resources are available, additional ECBs are created to perform more work.

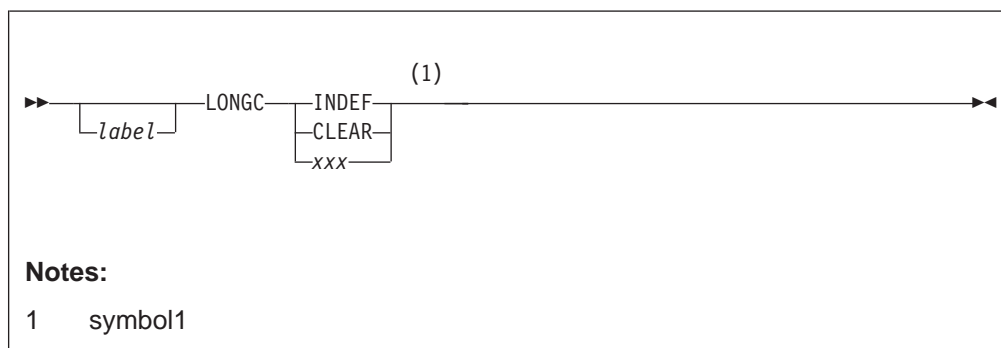
Create macros copy the LODIC characteristics to their child ECBs. This is why the LODIC macro with the UNMARK parameter is placed before the CREMC macro. The ECB that is being created for entry to BRFM will not have the low-priority of its creating ECB.

```
CHECK DS 0H
 LODIC SUSPEND,CLASS=BATCH,AVAIL=DOWRK
 B CHECK
DOWRK DS 0H
 LODIC UNMARK
 CREMC BRFM
 B CHECK
```

## LONGC–Set Entry Maximum Existence Time

This general macro allows an ECB-controlled program to set or reset the maximum allowed lifetime of the current entry in the system.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *symbol1*

This specifies the life-time for the ECB.

#### **INDEF**

ECB may exist indefinitely (long life entry detection will be inhibited for this entry).

#### **CLEAR**

Reset long life entry detection fields, the entry will be given a maximum life (from the time that the LONGC CLEAR was issued) of 1 minute.

#### *xxx*

From 1 to 254, set the maximum life time of the entry in minutes.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the instruction following the macro expansion.
- The condition code and general purpose registers will not be changed.

### Programming Considerations

- The LONGC macro is restricted to ECB-controlled programs.
- The ECB reference register (R9) must contain the address of the ECB issuing the LONGC macro.
- LONGC CLEAR or LONGC xxx require 8 bytes of storage.
- LONGC INDEF requires 4 bytes of storage.
- The long life entry detection program is inhibited for all entries that do not issue LONGC CLEAR or LONGC xxx.
- LONGC xxx will cause the long life entry detection program to flag the entry as “looping” if it is still active in the system approximately xxx + 1 minutes later.
- LONGC CLEAR is equivalent to LONGC 0 (LONGC 0 is invalid).

## LONGC

- LONGC CLEAR or LONGC xxx should be used to enable long life entry detection for applications that require this support, preferably by the application's input message editor.
- LONGC INDEF can be used by an application program to inhibit long life entry detection for a specific transaction, even though it has been enabled by a previous program.
- The long life entry detection program sends a message to the system operator whenever an entry is found to have exceeded its maximum life in the system.

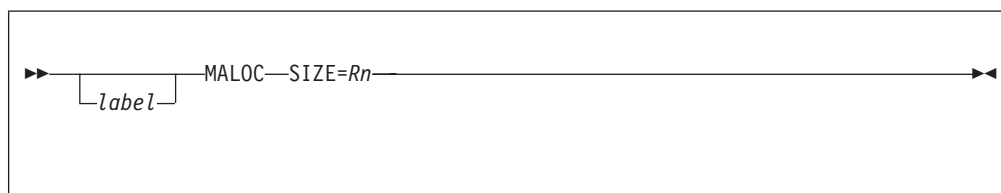
## Examples

None.

## MALOC—Reserve a Storage Block

This general macro obtains a variable-sized, doubleword-aligned block from heap-resident storage.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**SIZE=Rn**

A general register (R0–R7, R14, R15) or a register equate containing the number of contiguous bytes of storage to be allocated.

When coded in the control program (CP), only R0–R6 can be used.

### Entry Requirements

- The specified register contains the number of bytes requested.
- When called from the CP, the caller must be in the ECB virtual memory (EVM), in 31-bit mode addressing, R10 cannot be used as the base register, and R9 must point to the entry control block (ECB).

### Return Conditions

- The register specified by SIZE contains the address at the beginning of the allocated storage.

**Note:** The storage is allocated in the EVM as part of the total ECB-unique storage area available to the ECB by the MALOC macro and the `malloc C` function. It may be located above the 16-MB boundary. The address returned is valid in 31-bit addressing mode only.

- The return value of SIZE contains X'00000000' if:
  - SIZE is zero for input, or
  - The heap does not contain enough storage to satisfy the allocation request, and the heap cannot be extended further.

**Note:** If zero is returned and SIZE is greater than zero, the original storage block has not been released: its contents are still available to the caller.

- The contents of R14 and R15 are unknown. All other registers are preserved, except for any used by the SIZE parameter
- When called from the CP, the contents of register 7 are not preserved.
- Control is not returned if there is insufficient real storage to satisfy the request and a catastrophic system error results.
- Control is not returned if corruption is detected in the heap. A system error occurs, and the ECB exits immediately. See *Messages (System Error and Offline)* and *Messages (Online)* for more details about this error.
- Control is returned to the NSI if no heap corruption is detected.

## MALOC

### Programming Considerations

- Any valid register for an ECB-controlled program (R0–R7, R14, or R15) can be used for SIZE.
- Any valid CP register (from R0–R6) can be used for the SIZE parameter.
- When called from the CP, the contents of the ECB register save area (CE1S0x) are not preserved. For example, for SIZE=5, CE1S05 is not preserved.
- See also the `malloc` C language function and the `FREEC` and `RALOC` assembler macros for more information.
- In addition to the normal macro trace information the macro trace entry will contain the size, in number of bytes, and address of the allocated heap memory block. If there was no storage allocated, the address will be zero and the size will be the amount that was requested.
- This macro is primarily meant for E-type programs only. When it is called from the CP, it should not be from any SVC service routines, should be in the EVM, and should have an ECB pointer.

### Examples

This example shows the number of requested bytes that are being drawn from a DSECT and the address of the returned block that is being stored in an ECB location.

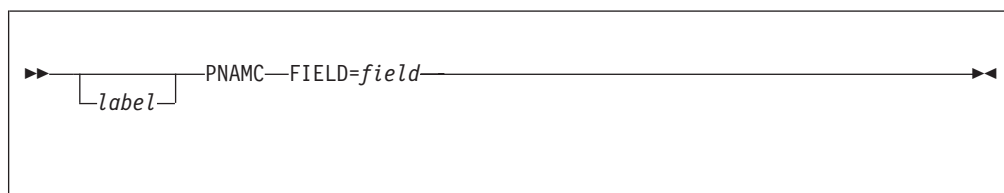
|                 |                                 |
|-----------------|---------------------------------|
| ITUUTL REG1=R14 | CONNECT WITH TABLE UPDATE DSECT |
| LA R14,ITULEN   | GET THE LENGTH OF A BLOCK       |
| MALOC SIZE=R14  | GET A STORAGE BLOCK             |
| ST R14,EBX000   | SAVE THE ADDRESS                |
| LTR R14,R14     | LOOK FOR RETURN CODE            |



## PNAMC—Move Program Name into Specified Field

This general macro is used to move the ECB-driven program name into the specified field from the program header.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**FIELD=field**

The field where the ECB program name is to be moved. The receiving field must be defined with a length attribute of 4.

### Entry Requirements

- R9 must contain a valid ECB address.
- R8 must contain the program base address.
- Addressability must be established to the specified field.

### Return Conditions

The effects of this macro are transparent: No registers are corrupted and the condition code remains unchanged.

### Programming Considerations

- This macro is to be used by E-type programs only.
- CZ4CP is invoked if it has not been called already.

### Examples

None.

You can use the EVNTC, POSTC, EVNWC, and SAWNC macros to pass the contents of a core block from one entry control block (ECB) to another ECB.

```

graph LR
 subgraph Line1 []
 direction LR
 P1[POSTC] --- T1[TYPE=]
 T1 --- CNT[CNT]
 T1 --- MSK[MSK]
 T1 --- CB[CB]
 T1 --- LIST[LIST]
 LIST --- B1[BLOCK=address]
 B1 --- L1[LEVEL=level]
 end

 subgraph Line2 []
 direction LR
 E2[ERCODE=] --- C2[character]
 C2 --- Rx[Rx]
 Rx --- N2[NFOUND=label]
 N2 --- L2[LINFND=label2]
 end

```

A symbolic name can be assigned to the macro statement.

The type of event being posted:

Post is posting a counter type event.

Post is posting a mask type event.

Post is posting a core block type event.

Post is posting a data list-type event.

**Note:** For TYPE=LIST, the BLOCK parameter is required.

The address of an area formatted as defined in EV0BK that contains the required POSTC input parameters. BLOCK and LEVEL are mutually exclusive except for TYPE=CB.

The core block reference word and the file address reference word (D0-DF) used to pass the information. LEVEL and BLOCK are mutually exclusive except for TYPE=CB.

**ERCODE=character(Rx)**

This parameter, if specified, indicates that this post is an error post. The error is specified as a 1-byte value or register notation, where the register contains the error code in byte 3.

For CNT-, MSK-, and CB-type events, the event is completed with an error return condition. The value of ERCODE is set in the error indicator field (BVNBKE) for the event.

For LIST-type events, the status flag field (EVNBKLIF) for the data item in error is set to indicate an error condition (X'40'). The value of ERCODE is saved in the error indicator field (EVNBKLIE) for the data item. The event is not completed unless all data items in the list have been set to completed (X'80') or in error (X'40').

**NFOUND=label**

Label in which to branch if the event is not outstanding.

**LINFND=label2**

Label in which to branch if a data item to post to is not found in the event list. The status flag field for the data item in the post list is set to X'20' to indicate that the data item was not found in the event list.

## Entry Requirements

- POSTC using the LEVEL parameter
  - The symbolic name of the event to be posted, which is 8 characters long, must be in CE1FAx, where x is the specified level.
  - For CNT type posts, the content of CE1CRx+2 is ORed into the return mask field of the event. This is called the POST MASK 2 field. The current value of the event will be automatically decremented by 1 when the POSTC macro is issued.
  - For MSK type posts, CE1CRx must contain a 16-bit mask, which is converted to a 1's COMPLEMENT, and ANDed against the mask used to create the event. The contents of CE1CRx+2 will be ORed into the POST MASK 2 field of the event.
  - For CB type posts, the contents of CE1CRx are moved directly to the event area and the level is marked available. For a type CB only, one POSTC is allowed. The event is marked satisfied with the first post.
- POSTC using the BLOCK parameter.
  - For CNT type posts, EVNBKM2 can contain a 16-bit mask which is ORed into the POST MASK 2 field of the event. The current value of the event will be automatically decremented by 1 when the POSTC macro is issued.
  - For MSK type posts, EVNBKM1 must contain a 16-bit mask to be converted to a 1's complement and ANDed against the mask value in the named event. EVNBKM2 can contain a 16-bit mask to be ORed into the POST MASK 2 field of the event.
  - For CB type posts, the LEVEL keyword must be used to inform the SVC processor of which core block level is to be used to satisfy the event.
  - For LIST-type posts, each POSTC macro (post list) data item will be posted to the corresponding item in the event list. When all items have been posted, the event is completed. EVNBKLC must contain the count of items in the data list, EVNBKLS must contain the size of a list item, and EVNBKLI must contain the list of data items.
- R9 must contain the address of the ECB being processed.

## POSTC

- When the POSTC macro is called in the control program, the following conditions are necessary:
  - Register R0 contains the error code if the ERCODE parameter is coded
  - Register R1 contains the address of the event macro communication block (EV0BK)
  - When the event is a core block (CB) type, the EVNBKCB field must contain the system virtual address (SVA) of the core block to be posted
  - Registers R14 and R15 are not preserved across the call
  - The Block and Level parameters are not coded
  - The NFOUND parameter is optional
  - The POSTC service routine is run in system virtual memory (SVM). The caller should be in SVM.

## Return Conditions

- Control is returned to the next sequential instruction and the condition code will be set accordingly.
  - Condition code EQ 0, event found and processed.
  - Condition code NE 0, event not found or data list item not found. For LIST-type events, the flag field in the data item also indicates not found (X'20').
- For TYPE=CB, the specified core block level will be marked as not holding a block.
- A system error will be declared, if the type of the post and the type of the event do not match. The posting ECB will be exited.
- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.
- This macro can be run on any I-stream.
- Defined events in MDBF systems are subsystem unique.

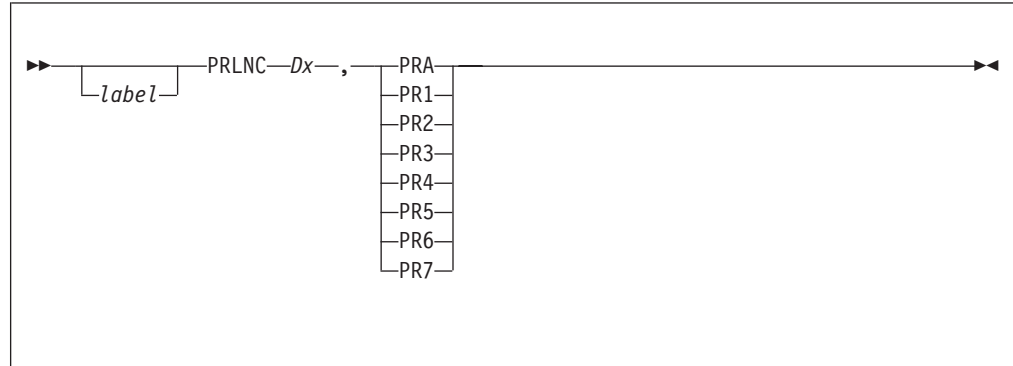
## Examples

None.

## PRLNC—Print a Line

This general macro prints one line of data from storage to the specified printer. The storage block referenced at the specified entry control block (ECB) data level is not returned to the storage pool when the I/O is completed. The assumed line length of 132 characters is printed.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

*Dx* A symbolic name (D0-DF) identifying the ECB data level that holds the data to be printed.

#### *prtdev*

A 3-character symbolic name for the printer must be specified (PR1–PR7, or PRA for any available printer). There is no default printer.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A storage block must be held by the ECB at the level specified.
- The application program must have previously issued a USURC macro requesting the assignment of the specified printer to this ECB.
- The printer control character (the CCW command code to be used) must be available in the 133rd byte of the data block following the header. The length of an output line is 132 bytes. The CCW command code can be found in UR0CMP.
- No other ECB should use this device.

## 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 condition code will be saved across this macro call.
- The program identification will be placed in the header of the data block by the service routine if the length of the header area in UR0IO is generated so that it is greater than 4 bytes.
- The file address reference word (FARW) at the specified level is unchanged.
- The storage block at the specified level is not available until the return from the next executed WAITC macro.

## PRLNC

- The status of the operation is unknown until the return from the WAITC macro. On a WAITC error return, the CE1SUD field for the specified level and CE1SUG are set as follows:
  - Bit 5 = 1 Means channel 12 has been detected.
  - Bit 6 = 1 Means the application program must abort.

## Programming Considerations

- This macro can be executed on any I-stream.
- To ensure completion of the print operation a WAITC macro must be issued.
- The service routine determines if the ECB is holding a proper size (not less than 381 bytes) storage block at the specified level and that the device is assigned to this ECB. If either condition is violated, the abort bit is set in the gross and detail error indicators (CE1SUG/CE1SUD) and the request is not serviced.
- The storage block containing the data is available to the operational program following return from the next executed WAITC macro.
- The valid command codes for printing are shown below.
  1. Write, Space 1 After Print.
  2. Write, Space 2 After Print.
  3. Write, Space 3 After Print.
  4. Write, Skip to Channel N After Print.

Where n can be 1 to 12. No other command codes are supported.

- The unit record support CSECT of the control program utilizes the data macro UR0IO, with the supplied storage address as a base, to set up the CCW for printing the line. The labels used are:

UR0HDR      The header area for this record. The length of this area is defined at system generation and may be from 0–24 bytes in length. The name of the program issuing this macro is placed in the program name field of the header area if the header definition is of sufficient size. Currently, a minimum of an 8-byte header size is required to save the program name.

UR0TXT      The text of the data to be printed. The length is 132 bytes.

UR0CMP      A 1-byte field containing the control character (CCW command code) for the printer.

All other areas of the storage block will not be used by the control program.

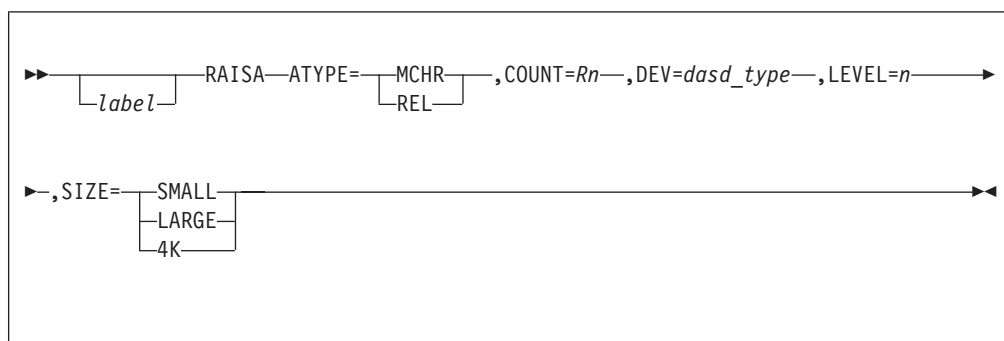
## Examples

None.

## RAISA—General File Get File Address

This general macro is used to increment the file address in the specified file address reference word (FARW) by the number of records indicated in the count register. In addition, for offline programs using general file relative record numbers, the relative record number is converted to a cylinder, track, and record number in a format suitable for channel command word usage.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ATYPE**

Specify one of the following:

##### **MCHR**

Indicates the file address is in the 4-byte MCHR format.

##### **REL**

Indicates the file address is a Relative Record Number.

#### **COUNT=Rn**

A register containing the number of records by which the file address is to be incremented. R14 and R15 must not be used.

#### **DEV=dasd\_type**

The TPF-supported DASD type on which the record resides (for example, 3380).

#### **LEVEL=n**

A number 0–F indicating the data level of the FARW.

#### **SIZE**

Specify one of the following:

##### **SMALL**

381-byte records.

##### **LARGE**

1055-byte records.

##### **4K**

4096-byte records.

## Entry Requirements

- For online and offline programs:
  - File address must be at a correct data level.
  - Count register must not be R14 or R15.

## RAISA

- A valid pseudo-module number must be used in the FARW.
- For offline programs:
  - One of the following must be coded to ensure addressability to the FARW.
  - R9 must be pointing to an area that is formatted as an ECB and macro EB0EB must be coded to address the ECB.
  - A fullword of storage within the offline program must be defined as EBCFAx (where x is the data level specified by LEVEL=).

## Return Conditions

- For online and offline programs:
  - Control is returned to the next sequential instruction.
  - The FARW at the specified data level contains the new file address.
  - The control bits in the record number field of the FARW will be set according to ATYPE and SIZE operands.
  - The condition code is unpredictable.
- For ATYPE=MCHR:
  - R14 will contain the head number of the file address.
  - R15 will contain the record number of the file address.
  - The register specified by COUNT= will contain zeros.
- For ATYPE=REL:
  - For an offline program call, the register specified by the COUNT parameter will contain the address of an 8-byte field with the following contents:

000CCHHR

Where:

- CC is the cylinder.
- HH is the track.
- R is the record.

## Programming Considerations

- This macro can be run on any I-stream.
- A General File Module Table must be link edited with ATYPE=REL offline programs. The pseudo module number is used to locate its entry in the General File Module Table. The base number is obtained from this entry to resolve the cylinder, track, and record address that is moved into the returned 8-byte field.
- The General File Module Table used by ATYPE=REL offline programs must match the table used online. This is to assure that the file addresses calculated by the offline program will match those calculated by the online program. The online General File Module Table is assembled in program segment CVZD. Either CVZD can be link edited with the offline program, or the DSECT Macros GFMTB and GFMTD (GENFD), can be assembled with the offline program. If the macros are used, the same parameters must be used that are used in CVZD.
- References are made to the FARW in offline programs that would normally be resolved without any consideration in online programs. See 'Entry Requirements' above, for coding techniques to establish addressability to the FARW in offline programs.



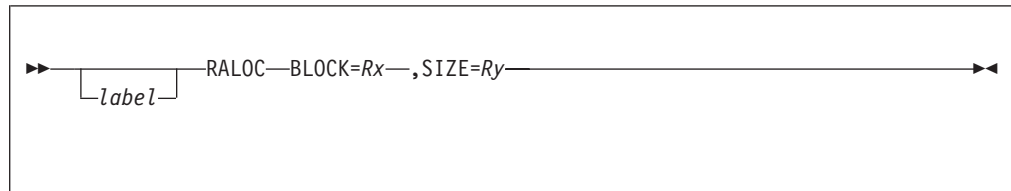
- The determination of an offline program is done by checking global symbol &BG0. Global symbol &BG0 is set to 1 when the BEGIN macro is assembled in an online program. It is assumed that offline programs will not use the BEGIN macro.

## Examples

None.

Use this macro to change the size of a block of storage allocated by a MALOC or CALOC macro.

## Format



A symbolic name can be assigned to the macro statement.

A general register (R0–R7, R14, or R15) containing the address of the block of storage to be resized, or zero.

A general register (R0-R7, R14, or R15) containing the number of contiguous bytes of storage to which the block is to be resized.

When coded from the CP only R0-R6 can be used.

## Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- The register specified by the BLOCK parameter must contain either the value returned by a previous CALOC, MALOC or RALOC macro (or by the C language function equivalents calloc, malloc or realloc), or zero.
- When called from the CP, the caller must be in the ECB virtual memory (EVM), in 31-bit addressing mode, R10 cannot be used as the base register, and R9 must point to the ECB.

## Return Conditions

- Control is returned to the next sequential instruction (NSI) if no system errors are detected.
- The register specified by BLOCK contains the address of the resized block of storage; if the new size is zero or if a resized block could not be allocated, the register contains zero. The data contents of the resized block are identical to the original block, up to the lesser of the size of the original block or SIZE. Any additional space beyond the size of the original block is uninitialized.

1. If the RALOC macro call causes an implicit growth to the heap, the BLOCK address may be different from what it was on entry. Pay particular attention to the possibility that zero may be returned (see the note that follows) and take steps to preserve the previous block's address to avoid a loss of data in the

block if a failure occurs. If the (nonzero) resized BLOCK address is different from the original BLOCK address, the original BLOCK is returned to the free storage pool of the heap.

2. If the RALOC macro call is made with a zero BLOCK address, the RALOC macro is processed as if it was a MALOC macro call for the specified size.
  3. If the RALOC macro call is made with a nonzero BLOCK address and a SIZE of zero, the BLOCK is released and zero is returned.
  4. The storage is allocated in the EVM as part of the total ECB-unique storage area available to the ECB. It is located above the 16-MB boundary. The address returned is valid in 31-bit addressing mode only.
- The register specified by SIZE is unchanged.
  - The return value, in the register specified for BLOCK, contains X'00000000' if:
    - SIZE is zero, or
    - the heap does not contain enough storage to satisfy the request for allocation, and the heap cannot be further extended.

Note that if zero is returned and SIZE is greater than zero, the original storage block has not been released; its contents are still available to the caller.

- The contents of R14 and R15 are unknown. All other registers are preserved, except for any used for the BLOCK parameter.
- When called from the control program (CP), the contents of register 7 are not preserved across this macro call.
- When main storage blocks are exhausted, a catastrophic system error results because there is not enough real storage to satisfy the request. Control is not returned.
- If corruption is detected in the heap, a system error is issued and the ECB exits immediately. Control is not returned. See *Messages (System Error and Offline)* and *Messages (Online)* for more details about this error.

## Programming Considerations

- Any valid nonscratch register for an ECB-controlled program (R0-R7, R14, or R15) can be used for BLOCK or SIZE.
- Any valid CP register (from R0-R6) can be used for the SIZE or BLOCK parameters.
- When called from the CP, the contents of the ECB register save area (CE1S0x) are not preserved; for example if SIZE=5, CE1S05 is not preserved.
- See also the `realloc` C function and the CALOC, FREEEC and MALOC assembler macros for more information.
- In addition to the normal macro trace information the macro trace entry will contain the size, in number of bytes, and address of the resized heap memory.

## Examples

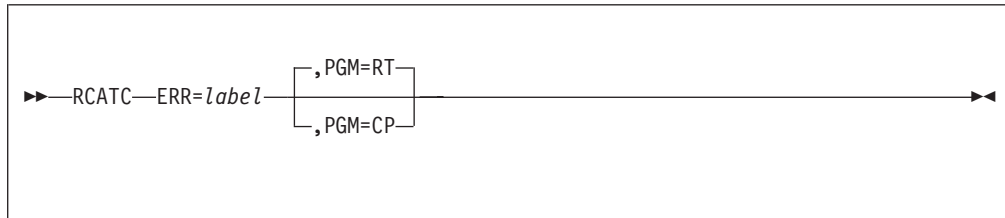
None.

## RCATC—Find an RCAT Entry

This general macro searches the routing control application table for an application name. The base address of the RCAT record containing the entry and the displacement to the entry in that record are returned for E-type programs. For C-type programs only, the RCAT entry address is returned.

If the application name is not found, the error return label receives control.

### Format



#### **ERR=label**

A required parameter specifying an error return label for use when an entry is not found.

#### **PGM**

An optional parameter where the valid values are:

##### **RT**

Used when the caller is an E-type (ECB) program. When PGM is not specified, this is the default.

##### **CP**

Used when the caller is a C-type (Control) program.

### Entry Requirements

- R0–R7 are reserved for use by this macro.
- R2 must contain the base address of the RCAT record (E-type programs only).
- R7 must contain the address of a 4-byte field where the application name of the RCAT entry to be located will be placed.
- R9 must contain the address of the ECB being processed (E-type programs only).

### Return Conditions

- Control is returned to the next sequential instruction when the entry is found.
- If the entry is not found, control is returned to the location specified by the user error return label.
- The contents of R0, R1, R3, R4, and R6 are unknown. The contents of all other registers are preserved across this macro call.
- For an E-type program, R2 will contain the base address of the RCAT record containing the entry and R5 will contain the displacement to the entry within the record.
- For a C-type program, R7 will contain the RCAT entry address.
- The USING statement for the RCAT will have been issued.

## Programming Considerations

- This macro can be run on any I-stream.
- An error routine must be defined in the user program to handle the error return.
- This macro can be called only once in any program.
- The macro first searches for the record containing the application name. When the record is found, a binary search is used to locate the application name in the record.

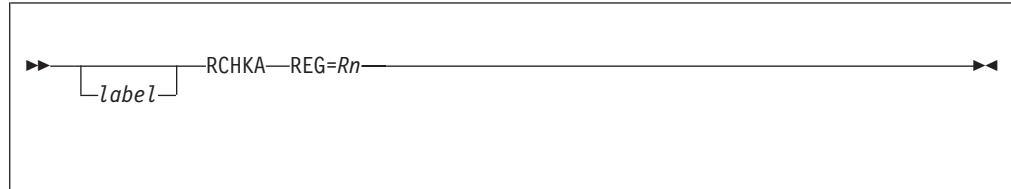
## Examples

None.

## RCHKA—Record Code Check

This general macro provides a variable value (1–254) for the CHK field of the data record header. This is done by using the global field @RCHKA, which is modified each time RCHKA is issued. The GLMOD and FILKW macros are issued in the macro expansion to allow modification of @RCHKA.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**REG=Rn**

Any application register (other than R0, R14, and R15) that will contain a variable value to be used as the record code check. This parameter must be coded.

### Entry Requirements

The GLOBZ macro must be issued before you issue RCHKA, and the base register must still be valid.

### Return Conditions

- Control is returned to the next sequential instruction.
- The new value of the record code check will be in the low order byte of the register. (R15 if parameter is omitted).
- The new value will also be stored in the Global Field @RCHKA+1.
- This macro changes the condition code.

### Programming Considerations

- This macro can be executed on any I-stream.
- The register specified as the REG parameter will be used and will contain the value of the record code check at macro completion. R0, R14, and R15 must not be used.
- The global field @RCHKA is updated each time this macro is issued.
- RCHKA uses the GLMOD and FILKW macros.

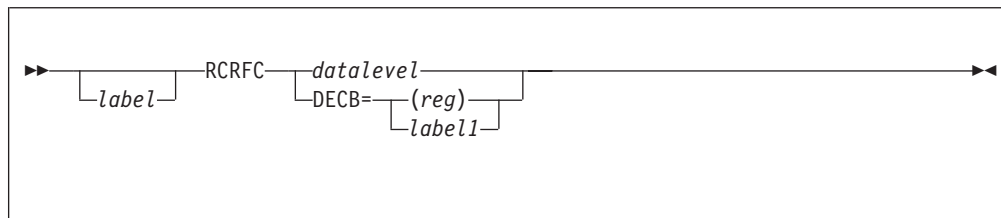
### Examples

None.

## RCRFC—Release Core Block and File Address

This general macro combines the functions of the RELCC and RELFC macros. The storage block specified by the designated core block reference word (CBRW) is released from the entry control block (ECB) and returned to the appropriate available working storage pool. The pool file address specified by the designated file address reference word (FARW) is returned to the available pool of file records. The macro service determines the record size and the file record address type (short-or long-term file storage).

### Format



*label*

A symbolic name can be assigned to the macro statement.

*datalevel*

The ECB data level (D0–DF) holding the storage block and file address to be returned.

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB) holding the storage block and file address to be returned.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A storage block must be held by the ECB on the specified ECB data level or DECB.
- A pool file address must be held by the ECB on the specified ECB data level or DECB.

### 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 CBRW is set to indicate that the storage block is no longer held.

### Programming Considerations

- This macro can be run on any I-stream.
- Separate invocations of this macro must be made for each storage block/pool address to be released.
- File pool support must be active when this macro is issued; the system must be above Utility State.
- The macro service checks the CBRW of the specified ECB data level or DECB to determine if a storage block is held. A system error dump is taken if a block is not held and the entry is exited.

## RCRFC

- Storage blocks should be released by the user as soon as possible.
- All pool addresses obtained by GETSC, GETLC, and GETFC must ultimately be released by a RELFC or RCRFC macro.
- No pool file address can be released more than once. A system error dump is taken if a block is released twice.
- TPF transaction services processing affects RCRFC macro processing in the following ways:
  - The file address is released only at commit time.
  - After the TXRBC macro has completed successfully, file address release requests are discarded and file addresses are not released.
  - If a system error occurs, processing ends as if a TXRBC 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

None.

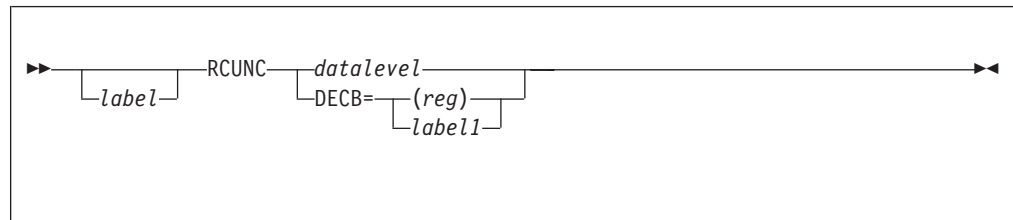


## RCUNC—Release Core Block and Unhold File Record

This general macro combines the functions of the RELCC and UNFRC macros. It releases the storage block referenced by the specified core block reference word (CBRW) from the entry control block (ECB) and returns it to the appropriate available storage pool. Then it releases the hold on the file record address referred to by the corresponding file address reference word (FARW).

The CBRW is made available for use, and the FARW is unchanged. If another entry has a request pending to find and hold the file record concerned, it is activated by this macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*datalevel*

The ECB data level (D0–DF) holding the storage block and file address to be unheld.

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB) holding the storage block and file address to be unheld.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A storage block must be held by the ECB at the specified ECB data level or DECB.
- A previously held file address must be contained in the FARW for the specified ECB data level or DECB.

### 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 CBRW is set to indicate that the storage block is no longer held.
- The FARW is unchanged.
- If another entry has a pending request to find and hold this file record, it is activated by this macro.
- When the RCUNC macro has completed successfully, the file address appears to be unheld from the program point of view. For requests from outside of the commit scope, the file address still appears to be held. If a request exists in the commit scope, it will be serviced. When commit processing has completed successfully, the file address is unheld and any waiting requests are serviced.

## RCUNC

- If you call the RCUNC macro outside a commit scope in a system that is not loosely coupled, the lock on the file address is immediately released. The lock can then be granted by the system to the next waiting ECB, if any.
- If you call the RCUNC macro outside a commit scope in a loosely coupled system, the lock on the file address is released when all previous DASD writes have been written to the media requested (virtual file access (VFA), DASD control unit cache, or the DASD surface). When the lock is released, the lock can then be granted by the system to the next waiting ECB, if any.
- If you call the UNFRC macro inside a commit scope (that is, after a tx\_begin function call is issued and the commit scope has not been ended by a tx\_commit or tx\_rollback function call), the lock on the file address is not released to the system. Instead, the lock becomes available for use by any ECB operating in the same or lower-level commit scope.

When the commit scope ends with a tx\_commit function call, all pending DASD writes are written to the media requested (VFA, DASD control unit cache, or the DASD surface). After the DASD writes are completed successfully, locks are released to the system. At this point the locks can be granted to other ECBs that are not in the commit scope.

When the commit scope ends with a tx\_rollback function, all pending DASD writes are discarded. The RCUNC macro works differently depending on whether the lock was acquired as part of the current commit scope or before the current commit scope. If the lock was acquired before the current commit scope, the RCUNC macro is ignored when the TXRBC macro is called. This leaves the lock held by the ECB, perhaps as part of a previous commit scope. If the lock was acquired during the current commit scope, both the acquiring of the lock and its release are rolled back and the lock is immediately available. If the commit scope being rolled back is nested inside a higher-level commit scope, the lock (hold) may be released to either the TPF system or to the higher-level commit scope depending on whether the lock was previously part of the higher-level commit scope.

- You cannot call the RCUNC macro on a record that is part of the suspended commit scope for an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FILEC record x
  3. TXSPC
  4. RCUNC record x.

## Programming Considerations

- This macro can be run on any I-stream.
- In a system generated with the HPO loosely coupled (L/C) function, the UNFRC portion of the macro service generates an I/O to release the hold on the requested file record.
- 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.

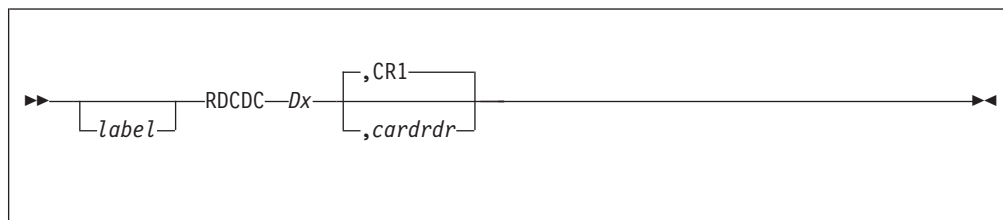
## Examples

None.

## RDCDC—Read a Card

This general macro causes the next card to be read into the specified storage block from the specified card reader. A 128-byte storage block is obtained by the service routine if no block is supplied by the application program.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *level*

A symbolic data level must be specified. Range D0–DF.

#### *cardrdr*

The 3-character symbolic name of the card reader must be specified for the available card reader. Specify one of the following:

- CR1
- CR2
- CR3
- CRA.

### Entry Requirements

- R9 must contain the address of the entry control block (ECB) being processed.
- No other ECB may be using the specified card reader.
- The USURC macro must have been previously called by this program requesting the assignment of the specified card reader to this ECB.
- If no storage block is held at the specified level, a 128-byte block will be retrieved by the service routine.

### 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 status of the Read operation is unknown. A WAITC macro must be executed to ensure completion of the operation.
- The file address reference word (FARW) at the specified level is unchanged.
- The contents of the core block reference word (CBRW) at the specified level is unknown.

### Programming Considerations

- This macro can be executed on any I-stream.
- The macro service ensures that the specified device is assigned to this ECB. If it is not, the abort switch is set on in the gross and detail error indicators in the ECB (CE1SUG/CE1SUD) and the request is not serviced.

## RDCDC

- To ensure completion of the Read operation, a WAITC macro must be issued before attempting to use the expected record. The storage block is not available until the WAITC completes. When control is returned to the operational program after the WAITC, the condition code and systems error/unusual condition indicators in the ECB will indicate the status of all completed I/O operations. For errors associated with the card reader, the CE1SUD for the specified level and CE1SUG are set as follows:
  - Bit 5 = 1, the logical end-of-file has been reached.  
(This is a card with C'/'\*\* in card-columns 1 and 2).
  - Bit 6 = 1, the application program must abort
- The user need not provide a CCW command code for this I/O operation. The macro service routine provides the command code of Read, Feed and Select Stacker 1.
- The macro service routine uses the data definition UR0IO to determine where the data read will be stored. Using the storage address supplied, or storage obtained by the service routine as the base of UR0IO, data is read into UR0TXT. The card will be read and stacked into stacker 1.

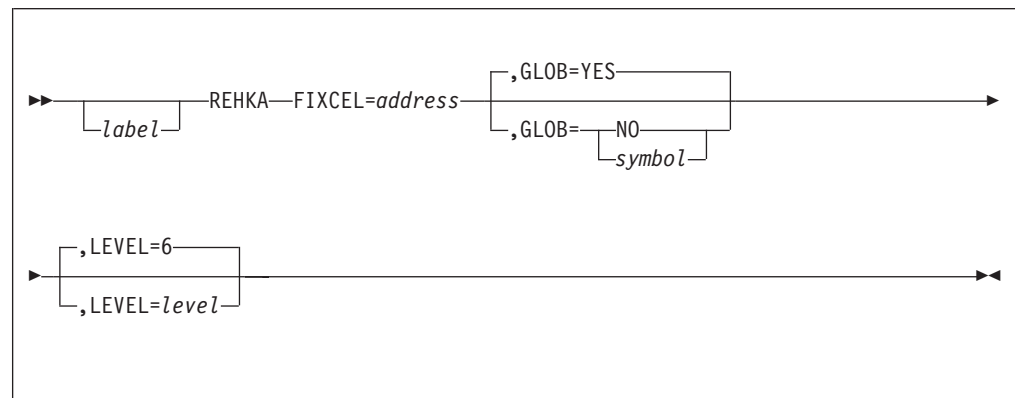
## Examples

None.

## REHKA—Rehook Core Block

This general macro transfers the information from a specified 8-byte field to a specified core block reference word (CBRW) in the ECB, and zeroes the 8-byte field. This allows access to the specified block by successive ECBs. If the field specified is in protected application or global storage, the GLMOD and KEYRC macros are used to allow modification of the specified field and restoration of the working storage protect key. The core block referenced by the 8-byte field must have been ***unhooked*** by using the UNHKA macro. The UNHKA and REHKA functions are designed for use by programs with unique buffering requirements.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FIXCEL**

This parameter is used to specify the address of the data field.

#### *address*

The operand 'address' must be the symbolic address of the 8-byte field that will be placed in the requested CBRW.

#### **GLOB**

The parameter specifies where the FIXCEL address resides in storage.

#### **YES**

YES means the specified FIXCEL area is in global area 1. This is the default if GLOB is not coded.

#### **NO**

NO means the specified FIXCEL area is not in protected application or global storage.

#### *symbol*

The operand 'symbol' must represent a protected application or global storage area corresponding to the FIXCEL area location. Valid symbols are GLOBAL1, GLOBAL2, GLOBAL3 and APL1.

#### **LEVEL**

This parameter is used to specify a core block level.

#### *literal*

A valid ECB data level (D0–DF). If not coded level 6 is assumed.

## REHKA

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The program must **not** be holding a core block on the level specified by LEVEL.
- The field referenced by the FIXCEL area must be 8 bytes.

### Return Conditions

- Control is returned to the next sequential instruction.
- The condition code is unchanged.
- The CBRW requested by LEVEL will contain the contents of the FIXCEL field.
- The FIXCEL field will contain zeros (0).

### Programming Considerations

- This macro can be executed on any I-stream.
- The use of this macro must be limited to programs with those unique buffering requirements that demand a specific block be capable of access by successive ECBs. It is **not** to be used to obtain use of a **free** data level.
- The program must ensure the FIXCEL area is assessable (for example, the correct address of storage protect key) if FIXCEL is in a protected area other than global storage.
- The CBRU specified by LEVEL= must not have an attached block.
- This macro may call the GLMOD and KEYRC macros.

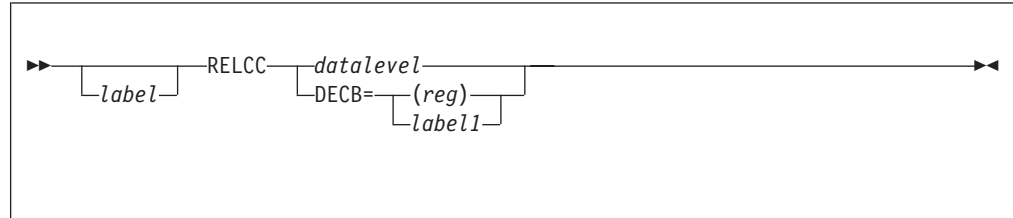
### Examples

- Before Execution  
CBRW4 (CE1CR4) contains X'001D4CA00001041F'  
  
@BRCPG contains X'000FEA000021017D'
- Execute  
REHKA FIXCEL=@BRCPG,LEVEL=4
- After Execution  
CBRW4 contains X'000FEA000021017D'  
  
@BRCPG contains XL8'00'

## RELCC—Release a Core Storage Block

This general macro releases from the entry control block (ECB) the storage block specified by the core block reference word (CBRW) for the designated ECB data level or data event control block (DECB) and returns it to the appropriate available working storage pool.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*datalevel*

A symbolic ECB data level (D0–DF).

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the DECB holding the storage block to be released.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- A storage block must be held by the ECB at the specified ECB data level or DECB.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of all user registers are preserved across this macro call.
- The specified CBRW is set to indicate the storage block is no longer held.

### Programming Considerations

- This macro can be run on any I-stream.
- Separate invocations of this macro must be made for each storage block to be released.
- The macro service checks the CBRW of the specified ECB data level or DECB to determine if a storage block is held. A system error dump is taken if a block is not held and the entry is exited.
- The macro service checks the CBRW of the specified ECB data level or DECB to determine if the block type is valid. A system error dump is taken if the block type is not valid and the entry is exited.
- If a double release of a storage block occurs then a system error will occur and the entry will be exited.
- If a release is attempted for an invalid block address then a system error will occur and the entry will be exited.

## RELCC

- 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

None.



## RELFC—Return File Pool Address

## RELFC

- When the TXRBC macro has completed successfully, file address release requests are discarded and file addresses are not released.
- If a system error occurs, processing ends as if a TXRBC macro was called.
- 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

RELFC D0

This invocation returns the record referred to by the file address reference word associated with data level D0.



## RELPC

**ERROR=***label*1

An optional symbolic name of a routine where control is passed, if an error occurs during RELPC processing. Register R15 contains an error code indicating the type of error.

## Entry Requirements

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 R14 and R15, except on an error condition when ERROR is coded, are unknown. The contents of all other registers are preserved across this macro call.
- If the program is no longer in use and if the program resided in a working storage block, then the block is returned and the program's Program Allocation Table (PAT) entry is reinitialized.
- If ERROR was coded and an error occurs, the indicated error routine is executed. Also, register R15 will contain an error indicator code. The following tags are generated by the RELPC macro and should be used for interrogating this error code:

**IG\_LS\_ERR**

The specified version of the program was not found.

**IG\_NF\_ERR**

The requested program was not found.

**IG\_PP\_ERR**

The specified version of the program was allocated as PRIVATE.

**IG\_SL\_ERR**

The SPECIAL lock indicator is not set.

**IG\_RT\_ERR**

A program retrieval error occurred.

## Programming Considerations

- This macro can be executed on any I-stream.
- If the program specified has not been previously locked, no action is taken and control returns to the next sequential instruction.
- This macro is restricted for use by E-type programs. C-type programs may issue GETPC to generate a DSECT for the RELPC parameter list.
- The SPECIAL lock indicator will only be turned off once. If a RELPC UNLOCK=SPECIAL is issued and the SPECIAL indicator is not set the IG\_SL\_ERR indicator will be returned. Normal RELPC requests ignore the SPECIAL indicator.

## Examples

None.



## RIDCC

RVT1, pseudo-LNIATA, and network addresses. If the input is an SCB2, compute SCBID and SCB1 instead of RID and RVT1.

### PLIT

Given a pseudo-LNIATA, check its correctness and calculate its RID and its SAT, RVT1, RVT2, and network addresses.

### NETA

Given a network address, check its correctness and calculate its RID or SCBID and its SAT, RVT1 or SCB1, RVT2 or SCB2, and pseudo-LNIATA addresses. Byte 1 of R15 contains the subarea and bytes 2–3 contain the element.

### RVT1=Rx

A register (R0–R7) that receives the RVT1 address. If the input parameter is an SCBID, *Rx* receives the SCB1 address.

### RVT2=Ry

A register (R0–R7) that receives the RVT2 address. If the input parameter is an SCBID, *Ry* receives the SCB2 address.

### SAT=Rz

A register (R0–R7) that receives the SAT address.

## Entry Requirements

R15 must contain an appropriate entry parameter to be validated and converted.

## Return Conditions

- Assuming normal return and **SVC=YES**, R15 points to a list of output data. All the required data should be copied into storage owned by the ECB or loaded into registers before issuing any macro that could cause the ECB to give up control, which could make the contents of the output data area indeterminate. This list is composed of 7 consecutive fullword fields. The contents of each are right-justified and are shown in Table 9.

Table 9. Output List

| Bytes | Label   | Content                                                                                                                             |
|-------|---------|-------------------------------------------------------------------------------------------------------------------------------------|
| 0     |         | Reserved; set to zero.                                                                                                              |
| 1–3   | RIDRID3 | 3-byte RID of the network addressable unit (NAU). If the input parameter is an SCBID, SCB1, or SCB2, RIDRID3 contains the SCBID.    |
| 1     |         | Reserved.                                                                                                                           |
| 2–3   | RIDRID  | 2-byte RID of the NAU.                                                                                                              |
| 4–7   | RIDRVT1 | Address of the RVT1 entry for the NAU. If the input parameter is an SCBID, SCB1, or SCB2, RIDRVT1 contains the address of the SCB1. |
| 8–11  | RIDRVT2 | Address of the RVT2 entry for the NAU. If the input parameter is an SCBID, SCB1, or SCB2, RIDRVT2 contains the address of the SCB2. |
| 12    |         | Reserved; set to zero.                                                                                                              |

Table 9. Output List (continued)

| Bytes | Label    | Content                                                                                                                                                                                     |
|-------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13–15 | RIDPLIT  | The pseudo-LNIATA for the NAU, if applicable.<br><b>Note:</b> Pseudo LNIATA exists only for the 3270-SDLC family of devices. If there is no pseudo LNIATA, zeros are returned in its place. |
| 16–19 | RIDSAT   | When nonzero, it is the SAT entry address for the NAU.                                                                                                                                      |
| 20–23 | RIDNETA  | Network address of NAU.                                                                                                                                                                     |
| 20–21 | RIDSUBA  | The subarea portion of the network address.                                                                                                                                                 |
| 22–23 | RIDELEM  | The element portion of the network address.                                                                                                                                                 |
| 24–25 |          | Reserved; set to zero.                                                                                                                                                                      |
| 26    | RIDSCBIN | SCB indicator. If the input parameter was an SCBID, SCB1, or SCB2, this byte is set to 1 (X'01'); otherwise, is set to zero.                                                                |
| 27    |          | Reserved; set to zero.                                                                                                                                                                      |

**Note:** Bytes 16–19 and 22–23 might be zero for a cross-domain resource, because the host knows about the network address of the NAU only after having established a session.

- If SVC=NO, the requested information is returned in the specified parameter registers. The contents of R14 indicate whether the returned data refers to an RVT or an SCB; if the input parameter was an SCBID, R14 contains one (X'00000001'); otherwise, R14 contains zero.
- If an error is detected, R15 contains zeros. The possible error conditions are:
  - SVC=NO or CODE=RID and the input parameter is neither a valid RID nor a valid SCBID.
  - CODE=RVT1 and the input parameter is neither a valid RVT1 nor a valid SCB1 address.
  - CODE=RVT2 and the input parameter is neither a valid RVT2 nor a valid SCB2 address.
  - CODE=PLIT and the input parameter is not a valid pseudo-LNIATA.
  - CODE=NETA and the input parameter is not a valid network address.
- If SVC=NO, R14 contains an indicator of whether the input parameter was a RID or an SCBID; if SVC=YES, the contents of register R14 are unpredictable. The contents of registers R0–R7 are preserved.

## Programming Considerations

Register R9 must contain the address of the ECB if the SVC=YES option is selected.

## Examples

None.

## RLCHA–Release Chain

This general macro releases the file address of a record and any records chained to it.

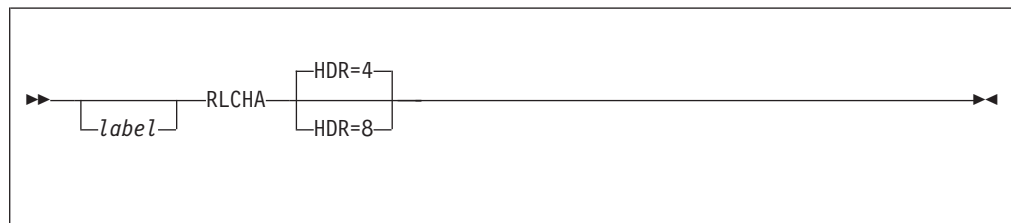
Given a file address, record id and RCC character it checks the record found at the address given by comparing the ID and RCC there for agreement with those given. If the record matches, it releases the file address.

If a forward chain is found, it attempts to find the chained record, and repeats the above procedure. The forward chain address field of the record contains F'0' if there is no chained record.

If a hardware error, an ID/RCC discrepancy or an invalid file address occur, it issues a system error.

This macro generates a CREMC macro. The CREMC macro may cause depletion of main core storage.

## Format



*label*

A symbolic name can be assigned to the macro statement.

### HDR=

The standard header. Valid values are:

- 4** Release the chain by using 4-byte file addresses.
- 8** Release the chain by using 8-byte file addresses.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- R14 must be available. RLCHA inserts byte count of the field referenced by R15.
- For HDR=4, R15 must contain the address of a 12-byte field formatted as follows:

### Bytes Contents

- 0–1 Record ID
- 2 Record code check
- 3–7 Not used
- 8–11 File address of first record to be released.

- For HDR=8, R15 must contain the address of a 24-byte field formatted as follows:

### Bytes Contents

- 0–1 Record ID



- 2        Record code check
- 3–7     Not used
- 8–11    Not used
- 12–15   Not used
- 16–23   8-byte file address of the first record to be released.

## 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 ECB levels remain unchanged.

## Programming Considerations

- This macro can be run on any I-stream.
- To be released, all succeeding records must have the same record ID and code check as the first record in the chain. If the succeeding records do not have the same ID and code check, the RLCH subroutine exits.
- This macro issues a CREMC to the RLCH program. The CREMC macro may cause depletion of storage.
- TPF uses register R14 which contains F'12' after the RLCHA macro runs.
- TPF transaction services processing affects RLCHA macro processing in the following ways:
  - The RLCHA macro is not called until commit processing has completed successfully.
  - When the TXRBC macro has completed successfully, the RLCHA requests are discarded and file addresses are not released.

## Examples

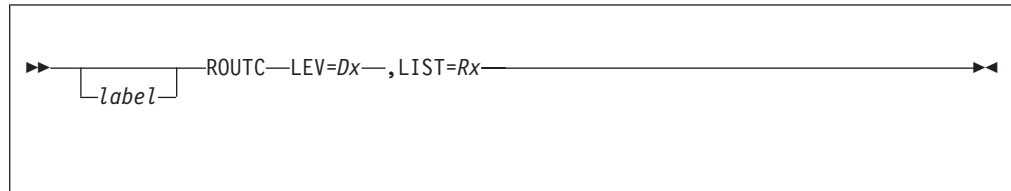
RLCHA

The RLCHA invocation uses the address in R15 to refer to a data field containing the record id, record code check, and file address of the record to be released. The record is released and any records chained to it are released as well.

## ROUTC—Route a Message

This general macro allows an application program to route a data message to a terminal or to another application program. The destination terminal or program can be hosted by the same processor as the originating application or by a different processor.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **LEV=Dx**

The argument required for this keyword parameter is the entry block data level (D0–DF), which contains the message block.

#### **LIST=Rx**

The argument required for this keyword parameter is a symbolic register (R0–R7 inclusive), which contains the address of the router control parameter list (RCPL).

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The first or only segment of the message to be routed must be contained in a main storage block attached at data level Dn. If the message is not completely contained in the main storage block, it is continued in one or more file storage blocks of the same size as the main storage block. For messages that are to be routed to a terminal, only 381-byte file storage blocks are allowed.
- A message to be routed to an application program must be in application message format.
- A message to be routed to a terminal may be in either application message format or in output message format. The format used is indicated in the router control parameter list (RCPL).
- The RCPL contains the routing information needed to describe the origin, destination, and control characteristics of the data message. The expanded RCPL is required for use with TPF/APPC support. Refer to macro RC0PL for details on the format.

### 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 core block reference word (CBRW) is initialized to indicate that a block of storage is no longer available.
- Some error conditions detected by the ROUTC macro may cause the issuing program to be forcibly exited. Such error conditions include:
  - No main storage block held on data level Dn

- Invalid message length
- Invalid parameters associated with message block and/or the router control parameter list (RCPL).

## **Programming Considerations**

- This macro can be executed on any I-stream.
- The ROUTC macro can create a new ECB and the routing function may proceed independently of the program that called the ROUTC macro.
- When a new ECB cannot be created due to low levels of storage blocks, processing is delayed through simulation of the DLAYC macro until blocks become available.
- A system error occurs if the calling ECB is in the commit scope because TPF transaction services processing does not support the ROUTC macro.

## **Examples**

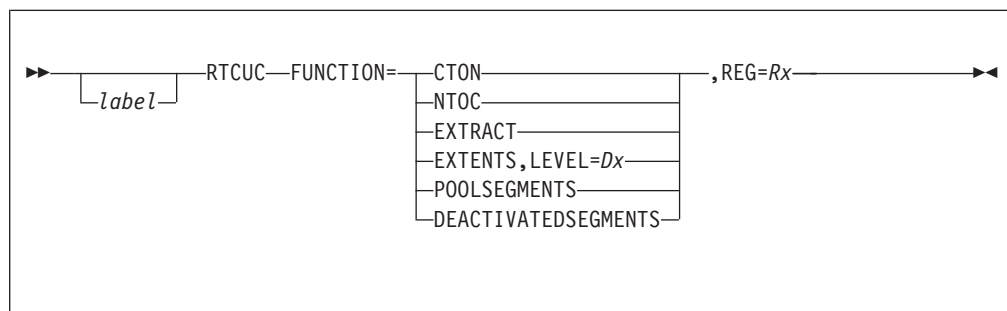
None.

## RTCUC–Record Type Conversion Utility

This general macro provides a centralized mechanism for interrogating the FACE table. Various functions are provided by the RTCUC macro:

1. Convert a fixed record type in its character representation of the SYSEQC tag to the corresponding numerical value.
2. Convert a fixed record type in its numerical form to the corresponding character representation of the SYSEQC tag.
3. Produce a list of extents for a particular fixed-type record.
4. Retrieve a fixed record type's characteristics from the FACE table.
5. Produce a list of segments for a particular pool section.
6. Produce a list of segments that are deactivated or will be deactivated for a particular pool section.

## Format



*label*

A symbolic name can be assigned to the macro statement.

### FUNCTION

The function to be performed from one of the following choices:

#### CTON

Convert character string to numerical value.

#### NTOC

Convert numerical value to character string.

#### EXTENTS

Produce list of extents.

#### EXTRACT

Retrieve fixed record type characteristics.

#### POOLSEGMENTS

Returns information about the current pool segment (PSEITK in DSECT IDSPS).

#### Notes:

1. To return information about the first pool segment, set PSEITK to zero.
2. To obtain information about all pool segments for a pool section, write a code loop that gets the next pool segment (PSENTK in DSECT IDSPS), saves it as the current pool segment (PSEITK), and issues RTCUC macro calls until the last pool segment is processed.

**DEACTIVATEDSEGMENTS**

Starting from the current pool segment (PSEITK in DSECT IDSPS), returns information about the first pool segment that is deactivated or pending deactivation.

**Note:** To obtain information about all pool segments that are deactivated or pending deactivation, write a code loop that gets the next pool segment (PSENTK in DSECT IDSPS), saves it as the current pool segment (PSEITK), and issues RTCUC macro calls until the last pool segment is processed.

**REG=Rx**

Must be specified as "REG=Rx", where Rx is in the range of R0–R7. This parameter is required.

For NTOC, EXTENTS, and EXTRACT the general register Rx must contain the numerical value of the record type.

For CTON, general register Rx must contain the address of an 8-byte left-justified blank padded field containing the character representation of the SYSEQC tag.

For POOLSEGMENTS, general register Rx must point to an area of memory defined by DSECT IDSPS.

For DEACTIVATEDSEGMENTS, general register Rx must point to an area of memory defined by DSECT IDSPS.

**LEVEL=Dx**

Valid only for the EXTENTS function and must be specified as "LEVEL=Dx." On return, the storage block on ECB data level Dx will contain the number of extents and the extents list. See the DCTRXTX DSECT for the format of the output block.

**Entry Requirements**

- R9 must contain the address of the ECB being processed.
- The RTCUC service routine is available to E-type programs only.

**Return Conditions**

No system errors are issued by the RTCUC macro service routine. Any invalid or illogical input parameters or conditions detected during processing are signaled to the caller by the condition code setting on return.

- When the condition code is 0

**CTON**

General register Rx, which contained the address of the record type on input (character string), will contain the numerical value of this record type on output.

**NTOC**

General register Rx contains the record type (numerical value) when RTCUC is called and contains the address of an 8-byte, left-justified, blank-padded character representation of the SYSEQC tag when RTCUC returns.

**EXTRACT**

General register Rx contains the record type when RTCUC is called and contains an output bit string representing the fixed record characteristics when RTCUC returns. This string is mapped by the IDSXTR DSECT and is described below.

|                                                     |       |                                                              |
|-----------------------------------------------------|-------|--------------------------------------------------------------|
| IXTRINF_X0 - Information Byte 3 (unique indicators) |       |                                                              |
| IXTRSSU                                             | X'80' | Indicates SSU unique (1) or SSU nonunique (0)                |
| IXTRPRC                                             | X'40' | Indicates processor unique (1) or processor nonunique (0)    |
| IXTRIST                                             | X'20' | Indicates I-stream unique (1) or I-stream nonunique (0)      |
| IXTRINF_X1 - Size index                             |       |                                                              |
| IXTRSM                                              | X'00' | Indicates a small record                                     |
| IXTRLG                                              | X'04' | Indicates a large record                                     |
| IXTR4K                                              | X'08' | Indicates a 4-K record                                       |
| IXTRINF_X2 - Record Characteristics                 |       |                                                              |
| IXTRCOM                                             | X'80' | Indicates shared (0) or unique (1) records                   |
| IXTRASZ                                             | X'40' | Indicates record size as 4 K (1) or not 4 K (0)              |
| IXTRPOOL                                            | X'20' | Indicates a fixed file record (0) or a pool record (1)       |
| IXTRVRT                                             | X'08' | Indicates a vertical record type                             |
| IXTRNRST                                            | X'04' | Indicates a record that cannot be restored                   |
| IXTRDUM                                             | X'01' | Indicates either a valid (1) or dummy (0) split chain header |
| IXTRREC_X3 - Information Byte 2                     |       |                                                              |
| IXTRLGE                                             | X'01' | Indicates a small (0) or a large or 4-K (1) record           |

### EXTENTS

ECB data level Dx will contain a count of the number of extents spanned by this record type, followed by a list of physical MMCCHHR start and end pairs for each extent, including duplicate extents if the record type is duplicated.

### POOLSEGMENTS

General register Rx will point to an area of storage that is defined by DSECT IDSPS. This area of storage will contain pool segment data.

### DEACTIVATEDSEGMENTS

General register Rx will point to an area of storage that is defined by DSECT IDSPS. This area of storage will contain data for a deactivated pool segment.

- When the condition code is 1

### CTON

Specified record type not found in the FCTB name table.

### NTOC

Invalid record type in general register Rx, either:

- Record type is less than 0.
- Record type is greater than the maximum value.
- No match found in FCTB name table.

### EXTRACT

Invalid record type in general register Rx, either

- Record type is less than 0.
- Record type is marked **not used** in the FACE table.

### EXTENTS

Invalid record type in general register Rx, either:

- Record type is less than 0.
- Record type is greater than the maximum value.

- Record type is flagged as 'not in use' in the FACE table.

**POOLSEGMENTS**

Pool section requested by general register Rx is not in use.

**DEACTIVATEDSEGMENTS**

Pool section requested by general register Rx is not in use.

- When the condition code is 2

**Function**

Description

**EXTRACT**

The record type in general register Rx exceeds the maximum record type defined in the system

**EXTENTS**

Data level or address specification problem of the following sort:

- R9 does not contain the address of an *in use* ECB.
- There is no storage block attached to the specified data level.
- The storage block supplied on data level Dx is not large enough to contain the extents list.

**POOLSEGMENTS**

Incorrect data passed by general register Rx, which is one of the following:

- Area of storage is not addressable.
- The PSEITK value in DSECT IDSPS is not valid.
- The record type detected in the FACE table (FCTB) is not valid.

**DEACTIVATEDSEGMENTS**

Incorrect data passed by general register Rx, which is one of the following:

- Area of storage is not addressable.
- The PSEITK value in DSECT IDSPS is not valid.
- The record type detected in the FACE table (FCTB) is not valid.

- When the condition code is 3.

**UNKNOWN**

The caller-specified function is not supported

- 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.

## Examples

- This example converts the character representation for the record type found in R5 to its numeric form.  
RTCUC FUNCTION=CTON,REG=R5
- This example converts the numeric representation for the record type found in R2 to its character form.  
RTCUC FUNCTION=NTOC,REG=R2
- This example provides a list of DASD extents associated with the record type found in R4 and return it in the block attached to data level DB.  
RTCUC FUNCTION=EXTENTS,REG=R4,LEVEL=DB

## RTCUC

- This example determines whether a record is a fixed or pool type. It does so by retrieving the characteristics of a record type specified in register R7 and using the 4-byte string of characteristics mapped by the IDSXTR DSECT to see if the record is a pool.

```

RTCUC FUNCTION=EXTRACT,REG=R7 GET RECORD CHARACTER
BC CC1,BDBF200 BAD RECORD TYPE, GET NEXT
ST R7,EBW040 SAVE RETURN VALUE
LA R7,EBW040 GET ADDRESS OF RETURN VALUE
PUSH USING
IDSXTR REG=R7 DSECT RESULTS FROM EXTRACT
TM IXTRINF_X2,IXTRPOOL IS THIS A POOL?
BO BDBF200 THEN GET NEXT RECORD TYPE
POP USING

```

- The following example obtains pool segment data for pool section SLT (small, long-term) on device type A.

```

IDSPS REG=R7
LA R7,EBX000 IDSPS is at EBX000
MVI PSERCC,#SLTA RCC for SLT on DEVA
XC PSEITK,PSEITK Zero input token to get 1st segment
AGAIN DS 0H
RTCUC FUNCTION=POOLSEGMENTS,REG=R7
BC CC1,SKIP If SLTA does not exist, skip
L R14,PSENTK Get next token
ST R14,PSEITK Save as input token
... Do the work
LTR R14,R14 Last pool segment?
BNZ AGAIN No, get next
...
SKIP DS 0H

```

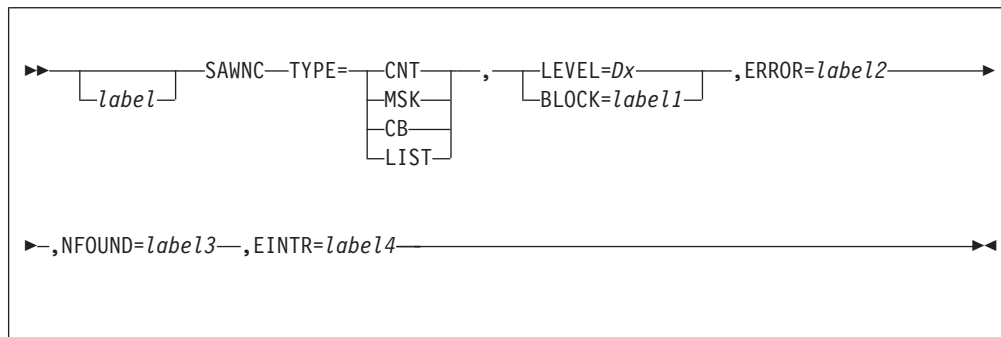


## SAWNC—Wait for Event Completion, Signal Aware

Use this general macro to wait for a named event to be completed. The SAWNC macro is similar to the EVNWC macro. However, if the caller of the SAWNC macro has to wait for an event to be completed, the caller can be interrupted by a signal. The SAWNC macro is used with the EVNTC and POSTC macros.

You can use the EVNTC, EVNWC, POSTC, and SAWNC macros to pass the contents of a core block from one entry control block (ECB) to another ECB.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### TYPE

The type of event:

##### CNT

Count event.

##### MSK

Mask event.

##### CB

Core block type of event.

If TYPE=CB is coded, the LEVEL keyword must be specified and the BLOCK keyword can also be specified. If the BLOCK keyword is specified, the event name is retrieved from the area specified by the BLOCK keyword, and the error code, if 1, is returned in the area specified by the BLOCK keyword.

##### LIST

Event is oriented around a corresponding list of specified values. Each POSTC macro that is issued contains a value that will be used to post the like value in the EVNTC specified list. When all values are posted in the event list, the event is completed.

**Note:** For TYPE=LIST, the BLOCK parameter must be specified.

#### LEVEL=Dx

The data level of a core block reference word (CBRW) and file address reference word (FARW). Valid values are D0–DF. The FARW contains the symbolic name of the event. The LEVEL and BLOCK keywords are mutually exclusive except when CB is specified for the TYPE keyword.

## SAWNC

### **BLOCK=***label1*

An area formatted as defined by the EV0BK DSECT that contains the name of the event. Other fields defined by the EV0BK DSECT are used to return values from the completed event. The LEVEL and BLOCK keywords are mutually exclusive except when CB is specified for the TYPE keyword.

### **ERROR=***label2*

A label where control is passed if the event ends with an error.

### **NFOUND=***label3*

A label where control is passed if the event does not exist.

### **EINTR=***label4*

The label of an instruction to be given control if SAWNC macro processing is interrupted by a signal.

## Entry Requirements

- Register 9 (R9) must contain the address of the ECB being processed.
- If only the BLOCK keyword is specified, the symbolic name of the event must be in the EVNBKN field in the area pointed to by the BLOCK keyword. For TYPE=LIST, allocate the full size of the EV0BK DSECT to ensure that there is enough room for the returned data list.
- If only the LEVEL keyword is specified, the symbolic name of the event must be in file address reference word CE1FAx, where x is the data level (0–F).
- If CB is specified for the TYPE keyword and the BLOCK keyword is also specified, the event name is retrieved from the area specified by BLOCK. Otherwise, the event name must be specified in the FARW specified by the LEVEL keyword.

## Return Conditions

- If only the BLOCK keyword is specified:
  - The area specified by the BLOCK keyword is filled as it relates to the specified event type. See “POSTC–Mark Event Completion” on page 306 for more information about the POSTC macro.
    - If it is a MSK type, the specified area will be filled in with the remaining mask and the contents of the accumulated POST MASK 2 field (EVNBkm2) from the event.
    - If it is a CNT type, the specified area will be filled in with the remaining count value.
    - If it is a CB type, the core block reference word (CBRW) specified by the LEVEL parameter is set to the value passed by the POSTC macro. Issue the LEVTA macro to determine if a core block was passed.
    - If it is a LIST type, the area specified will be filled in with the data list information. To determine the status of the data list items, the flag field (EVNBKLIF) of each item must be interrogated. Valid values for this flag field are defined in the EV0BK DSECT.
  - The EVNBKE field is set with the error indicator if an error occurred.
    - For the MSK-, CNT-, or CB-type event, the error indicator in the EVNBKE field is the value returned in the ERCODE parameter as coded in the POSTC macro.
    - For the LIST-type event, the error indicator in the EVNBKE field is set to a generic error code value of X'7F'. The error indicator field (EVNBKLIE) for each list data item contains the value returned in the ERCODE parameter as coded in the POSTC macro.

- If only the LEVEL keyword is specified, the CBRW for the level is set to the remaining mask or count value in CE1CRx bytes 0–1, where x is a data level (0–F), and to the contents of the accumulated POST MASK 2 field in CE1CRx bytes 3–4, where x is a data level (0–F).
- If CB is specified for the TYPE keyword, the CBRW and FARW specified are set to the values passed by the POSTC macro. Use the LEVTA macro to determine if a core block is passed.
- Control is returned to the next sequential instruction (NSI) except for the following conditions:
  - If the event named by either the BLOCK or LEVEL keyword is not found, an immediate return is made to the label specified by the NFOUND keyword.
  - If the event ended with an error, control is returned to the label specified by the ERROR keyword.
  - If SAWNC macro processing is interrupted by a signal, control is returned to the label specified by the EINTR keyword.
- The contents of R14 are unknown. The contents of R0–R7 are preserved across this macro call.
- The error indicated by R15 is returned in the ERCODE keyword as coded in the POSTC macro.

## Programming Considerations

- This macro can run on any I-stream.
- The SAWNC macro enables the named event to allow event completion to be posted to the ECB:
  - If the event has not completed successfully when SAWNC macro processing begins, the input/output (I/O) counter and flags for the named event are incremented so that when the event is completed, the event can be posted to the calling ECB.
  - If the named event has completed successfully when SAWNC macro processing begins, SAWNC macro processing starts the posting process.
- Defined events in multiple database function (MDBF) systems are subsystem unique.
- Using the SAWNC macro causes an unconditional loss of control for the calling ECB unless the named event is not found.
- More than one ECB can wait for the same named event except for a core block type of event. When the event is completed successfully, all waiting ECBs are posted as completed.
- For a core block type of event, only the creating ECB can wait on the event. For any other ECB attempting to wait on the event, control is returned to the label specified by the NFOUND keyword as a not-found condition.
- The EVNTC and SAWNC macros can be coded to delay an ECB for a specified number of seconds. This essentially delays the ECB by the amount of time specified by the TIMEOUT keyword of the EVNTC macro. It is not necessary to code the POSTC macro.
- Unless the SAWNC macro returns control to the calling ECB because the event name cannot be found, the calling ECB gives up control and subsequently regains control when one of the following occurs:
  - Event completion is posted (with or without error).
  - The timeout value ends.
  - A pending alarm is triggered, causing a SIGALRM signal to be sent.

## SAWNC

- A signal is sent to the calling process.

## Examples

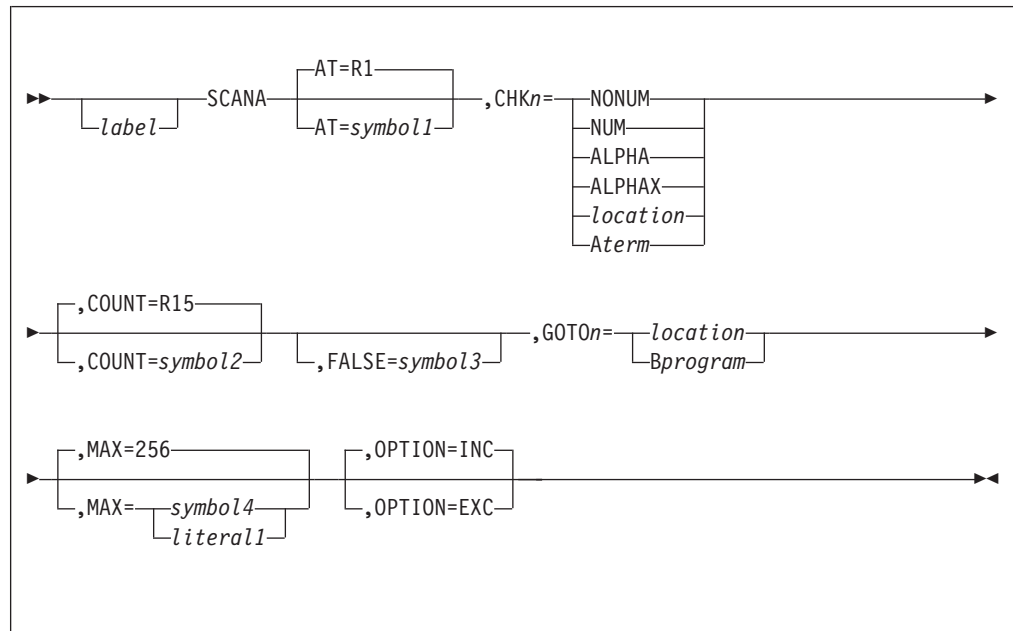
None.

## SCANA—Scan Data Field

This general macro searches a data field for the presence of any specified characters. A match with one of these specified characters will result in a branch to a specified location, or an ENTNC to any specified program. The user specifies the field to be searched and up to 9 characters, which will be used for matching arguments.

Associated with each of the 9 possible characters to be searched for is an option relating to the performance of the macro when a match is found. The count of bytes may either include the character that caused the match, or it may not.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**AT=symbol1|R1**

The symbolic name of an application register other than R0 referring to the first byte of a data field to be scanned. R1 will be assumed if the parameter is omitted.

**CHK<sub>n</sub>**

Characters to be used for matching with bytes in the input field. Each CHK parameter must have a corresponding GOTO parameter. The CHK parameters may be any of the following:

**NONUM**

Any nonnumeric character (that is, not digits 0–9).

**NUM**

Any numeric character (that is, digits 0–9) will cause a match.

**ALPHA**

Any alpha character (that is, A–Z) will cause a match. Consider use of the ALPHA macro instead of this action.

## ALPHAX

Any true alpha character (that is, A–Z, but excluding the nonalpha characters in the gap between EBCDIC I and J, and in the gap between EBCDIC R and S) will cause a match.

### *location*

Any byte location that will, at macro execution time, contain a character to be used for matching.

### *Aterm*

An **A** followed by any character or hexadecimal self-defining value that can be coded in a CLI instruction.

Examples are:

CHK1 = AC'B (defines a B character).

CHK1 = AX'1C' (defines a hexadecimal value of 1C).

## COUNT=symbol2|R15

The symbolic name of an application register other than R0 which will contain, at completion of the macro, the number of bytes including the byte that caused the macro to stop scanning, or if OPTION is EXC, the number of bytes preceding the byte which caused the macro to stop scanning.

R15 is assumed if the parameter is omitted.

If parameter MAX is a 1, code manipulating this register is not included since there is no need for counting.

## FALSE=symbol3

A symbolic location that will be branched to if no match is found within the data field. If omitted, control will be returned to the next sequential instruction (NSI).

## GOTO<sub>n</sub>

An action to be performed if a match occurs for the CHK1...CHK9 characters. The GOTO parameters are defined as follows:

### *location*

This is a location where the program branches if a match is found with the corresponding CHKn parameter. (The symbol must not start with a B).

### *Bprogram*

This is a 4 character program name preceded by a B. The program is entered with no return on a match with the corresponding CHKn parameter.

## MAX

The maximum number of bytes in the data field that are to be scanned or a reference to a register that contains the maximum. The default maximum is 256. The maximum for the MAX parameter is 65 536.

### *symbol4*

A register containing the maximum number of bytes (meeting the requirements of literal1). If the maximum number of bytes is equal to 1, symbol2 will not be used, since there will be no need to keep a count of the number of bytes scanned.

### *literal1*

Any number defined by the limits 1 <= literal1 <= 256. If equal to 1, COUNT parameter is ignored.

## OPTION

Specify one of the following:

**INC**

Causes the count of bytes to include the character that caused the match.

**EXC**

Causes the count of bytes to exclude the character that caused the match.

- The field to be scanned must be referenced by *symbol1*.
- If the CHK parameter uses a symbolic location, that location must contain the character to be used in the match at execution time.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- The contents of R14 and R15 are unknown. Only the registers specified in the macro call are used by the macro:

*symbol1*

At macro initialization, will contain a reference to the first byte of the field to be scanned. At macro completion, *symbol1* will point to the character that caused a match; or if no match occurred, *symbol1* will point to the beginning of the field + MAX.

*symbol2*

The count register is initialized to zero, and at macro completion contains the number of characters scanned as specified by the OPTION PARAMETER. If MAX=1, this parameter is not used.

*symbol3*

If used, the register specified containing the maximum number of bytes to be scanned will be unchanged at macro completion.

- Control is returned on a match according to the GOTO parameters. If MAX is reached before a match, and (*symbol3*) is used, control is returned to the location specified by (*symbol3*); if no (*symbol3*) is used, control is returned to the next sequential instruction.
- (*symbol2*) contains the number of characters scanned as specified by the OPTION Parameter. If OPTION is INC, *symbol2* will contain the number of bytes including the byte which caused the macro to stop scanning, or if OPTION is EXC, the number of bytes preceding the byte that caused the macro to stop scanning. If MAX is a 1, this parameter is not used.
- The scanned field is not altered.
- (*symbol1*) points to the character that caused a match.
- The condition code is changed.

## Programming Considerations

- This macro can be executed on any I-stream.
- When the maximum number to be scanned has been reached, and no match has been found, the FALSE parameter will indicate where control will be returned. If no FALSE parameter is used, control is returned to the next sequential instruction.
- The CHKi parameters will be processed in numerical order (that is, CHK1, CHK2, and so on). Processing in numerical order can be used to advantage with the ALPHA, ALPHAX, NONUM, or NUM options. For example, the NONUM option will cause a match on special characters as well as alphabetic characters. If a slash, an A, and NONUM were desired as separate options the code would be:

## SCANA

```
SCANA CHK1 = AC/,GOTO1 = PCA01,
 CHK2 = ACA,GOTO2 = PCA02,
 CHK3 = NONUM,GOTO3 = PCA03
```

If CHK1 was set as NONUM, the A and / would be treated as nonnumeric and no distinction would be achieved.

- When searching for an ALPHA character only, consider using the ALPHA macro.

## Examples

- The following example defines a scan starting at a location pointed to by R14 for a byte at location BXCMD1. The number of bytes scanned (and the match byte) reaching the match character is placed in R7. If a match is found, control transfers to location CBXM00, but if a match isn't found, control transfers to location BCXMERO.

```
SCANA AT=R14,COUNT=R7,CHK1=BCXMD1,GOTO1=CBXM00,FALSE=BCXMERO
```

- The following example defines a scan starting at the address in R1 for a either a slash character (/) or a non-numeric character. If a slash is found, control transfers to location CIPF160, but if a non-numeric character is found, control transfers to location GIPFMR08. A maximum of four characters is scanned and the number of characters scanned before a match is found is placed in R15.

```
SCANA MAX=4,CHK1=AC'/' ,GOTO1=CIPF160,CHK2=NONUM,
 GOTO2=GIPFMR08,OPTION=EXC
```

- The following example defines a scan starting at the address in R6 for a maximum of bytes indicated in R3. These bytes are scanned for either a X'15', a X'01' or any true alphabetic character. If X'15' is found, control transfers to location CIM130. If X'01' is found, control transfers to location CIM160, and if any exclusive alphabetic character is found, control transfers to CIM180. If no match is found, control resumes at the NSI. The number of characters scanned (including the match byte) is placed in R15 at the conclusion of the scan.

```
SCANA AT=R6,COUNT=R15,FALSE=CIM140,MAX=R3,CHK1=AX'15',
 GOTO1=CIM130,CHK2=AX'01',GOTO2=CIM160,
 CHK3=ALPHAX,GOTO3=CIM180
```

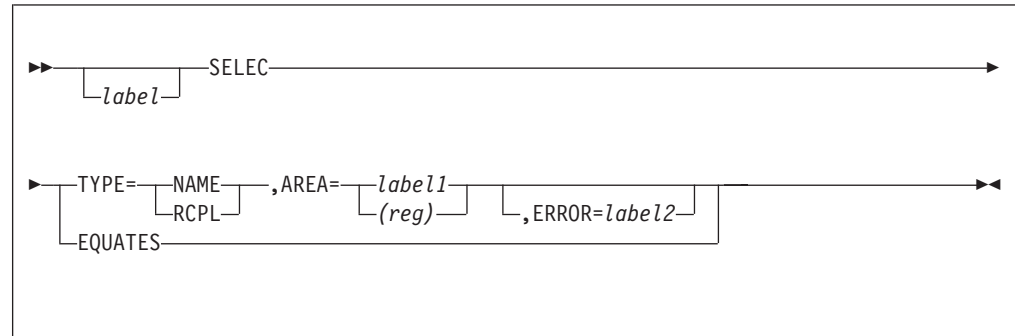


## SELEC—Select a Thread Application Interface

This general macro obtains information an application program requests. The information returned includes:

- RCPL
- Session partner.

### Format



#### EQUATES

This indicates that only a list of equates are generated and no executable code is generated. This is an optional parameter.

#### TYPE

indicates the kind of data being located.

#### NAME

A name is to be used as a search argument to locate the data. The NAME must be a network qualified name. Blanks may be specified for the network qualifier.

#### RCPL

This indicates an RCPL is to be used as a search argument to locate the data.

Refer to the Entry Requirements that follow.

#### AREA

indicates the location of the work area for input and output.

#### label1

A label that indicates the address of a work area where the input data can be found and that will be used to return the output.

#### (reg)

A register (R0–R6) that contains the address of a work area where input data can be found and that will be used to return the output.

#### ERROR

Specified an error processing location.

#### label2

A label to branch to if the an error is detected by the select-a-thread routine.

### Entry Requirements

- Register (R7) must contain the address of a user-supplied core block for SELEC macro usage.

## SELEC

- The format of AREA data input to SELEC when TYPE=NAME

| Offset Decimal | Offset Hex | Length | Description                           |
|----------------|------------|--------|---------------------------------------|
| 0              | 0          | 8      | Name of origin application.           |
| 8              | 8          | 8      | Netid of destination application.     |
| 16             | 10         | 8      | Node name of destination application. |
| 24             | 18         | 2      | Reserved.                             |
| 26             | 1A         | 1      | Numeric SLU qualifier.                |
| 27             | 1B         | 1      | Reserved.                             |

- The format of AREA data input to SELEC when TYPE=RCPL

| Offset Decimal | Offset Hex | Length | Description      |
|----------------|------------|--------|------------------|
| 0              | 0          | 16     | User input RCPL. |
| 16             | 10         | 12     | Reserved.        |

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of the scratch registers (R14, R15) are unknown.
- The contents of the remaining operational registers are saved while running this macro.
- This is the format of data returned by SELEC for TYPE=NAME

| Offset Decimal | Offset Hex | Length | Description                      |
|----------------|------------|--------|----------------------------------|
| 0              | 0          | 16     | SELCPL - Requested RCPL.         |
| 16             | 10         | 10     | Reserved.                        |
| 26             | 1A         | 1      | SELQUAL - Numeric SLU qualifier. |
| 27             | 1B         | 1      | SELRNCD - Error return code.     |

- This is the format of data returned by SELEC for TYPE=RCPL

| Offset Decimal | Offset Hex | Length | Description                                     |
|----------------|------------|--------|-------------------------------------------------|
| 0              | 0          | 8      | SELORIG - Name of origin application.           |
| 8              | 8          | 8      | SELID - Netid of destination application.       |
| 16             | 10         | 8      | SELNAME - Node name of destination application. |
| 24             | 18         | 2      | Reserved.                                       |
| 26             | 1A         | 1      | SELQUAL - Numeric SLU qualifier.                |
| 27             | 1B         | 1      | SELRNCD - Error return code.                    |

- Error return codes:
  - X'04'** Invalid name; Bad return from INQRC macro.
  - X'08'** Invalid request.
    - SLU not in session with PLU.

- Not a PLU/SLU combination.
- X'0C'** RIDCC error.
- X'10'** SLU not bound/not in session.
- X'14'** Invalid request.
  - Org/dest. not cross-domain environment.
  - Org/dest. not an LU.
- X'20'** SLU not active.
- X'24'** Discrepancy between input qualifier and input SLU.
- X'28'** Qualifier invalid.
  - Does not point to valid SLU.

## Programming Considerations

This macro can be executed on any I-stream.

## Examples

### 1. SELEC TYPE=NAME,AREA=EBW000,ERROR=ROUTINE

#### • INPUT

|                 |   |                                   |
|-----------------|---|-----------------------------------|
| EBW000 = 'CCCC' | ' | Origin application name           |
| EBW008 = 'NET1' | ' | Destination application netid     |
| EBW016 = 'CISB' | ' | Destination application node name |
| EBW026 = X'00'  |   | Numeric qualifier                 |

A core block must be attached to the ECB and pointed by R7.

#### • OUTPUT: RCPL is defined in RC0PL

|                    |                         |
|--------------------|-------------------------|
| EBW000 = X'000072' | Destination RID         |
| EBW003 = X'C1'     | Destination CUID        |
| EBW004 = C'CCCC'   | Origin application name |
| EBW005 = X'42'     | Control byte 0          |
| EBW0026 = X'01'    | Numeric SLU Qualifier   |
| EBW0027 = X'00'    | Return code             |

### 2. SELEC TYPE=NAME,AREA=(R1),ERROR=ROUTINE

R1 contains the address of the input and output work area.

A core block must be attached to the ECB and pointed by R7.

### 3. SELEC EQUATES

#### • INPUT

A core block must be attached to the ECB and pointed by R7.

#### • OUTPUT

|                            |                      |
|----------------------------|----------------------|
| SELCPL EQU 0,16            | RCPL                 |
| SELDESS EQU 0,4            | DEST SEQ #           |
| SELDES3 EQU SELDESS,3      | DEST RID (3-byte)    |
| SELDESR EQU SELDES3+1,2    | DEST RID (2-byte)    |
| SELDESP EQU SELDESR+2,1    | DEST CUID            |
| SELORG EQU SELDESP+1,4     | ORIGIN RESOURCE NAME |
| SELCTL0 EQU SELORG+4,1     | CONTROL BYTE 0       |
| SELRESVA EQU SELCTL0+1,7   | RESERVED AREA        |
| SELRESVB EQU SELRESVA+7,10 | RESERVED AREA        |
| SELQUAL EQU SELRESVB+10,1  | NUMERIC QUALIFIER    |
| SELRTNCD EQU SELQUAL+1,1   | RETURN CODE          |
| *                          |                      |
| SELORIG EQU 0,8            | ORIGIN RESOURCE NAME |

## SELEC

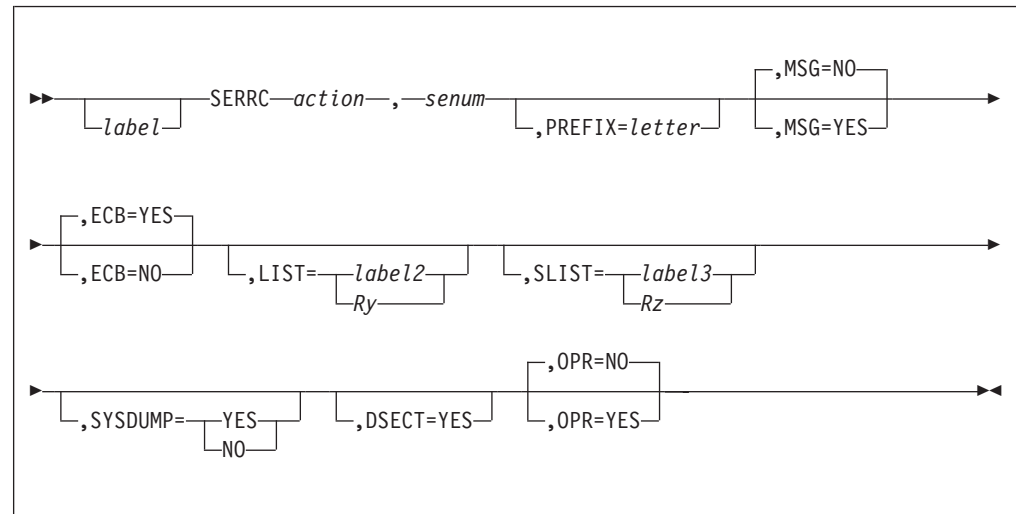
|         |     |             |               |
|---------|-----|-------------|---------------|
| SELID   | EQU | SELORIG+8,8 | NETID         |
| SELNAME | EQU | SELID+8,8   | RESOURCE NAME |
| SELREGS | EQU | 80,28       | REG SAVE AREA |

## SERRC—System Error

This general macro causes a storage dump and sends a message to the system operator via a CRAS terminal. It may be used whenever an operational program or the control program detects an abnormal condition that requires a dump of main storage as an aid to error analysis.

**Note:** All system activity is suspended while the storage dump is written. If the cause of the error is well understood and less than 32KB of data is required to fully document the problem, use the SNAPC macro. The SNAPC macro uses far fewer processor resources; in particular, system activity is not halted during its processing.

### Format



#### *label*

A symbolic name may be assigned to the macro statement.

#### *action*

Specifies the action to be taken by the system error routine upon completion of the dump:

- E** Exit the entry control block (ECB) if one is associated with the error. Otherwise, return to the CPU Loop.
- R** Return to next sequential instruction.
- C** Catastrophic error, re-IPL the system.

#### *senum*

Specifies the 6-digit hexadecimal system error number (range: X'000001' to X'FFFFFF') used to identify the dump. TPF system error numbers are documented in *Messages (System Error and Offline)* and *Messages (Online)*. Code this parameter in one of the following ways:

##### *hexnum*

A hexadecimal system error number in the range X'000001' to X'FFFFFF'.

##### *(errnum)*

A symbolic system error number, as defined in CZ1SE (IBM error numbers), enclosed in parentheses.

*Rx* The symbolic name of a register containing the system error number. Valid

## SERRC

symbols are R0 through R7, R14, and R15. If MSG=YES is coded, then Rx must not specify R0. Rx must not specify the same register as Ry or Rz.

### **PREFIX=***letter*

The prefix is an uppercase alphabetic character that is concatenated with the system error number in the console message and in the dump. The letters I and W–Z are reserved for IBM use. The default is PREFIX=I for IBM code, and PREFIX=U for non-IBM code.

### **MSG**

Specifies one of the following:

#### **YES**

R0 must contain the address of a user supplied message that is to be appended to the operator notification of the dump.

#### **NO**

Specifies there is no user supplied message.

### **ECB**

Specifies whether or not an ECB is associated with this error.

#### **YES**

An ECB is associated with this error. Dump the ECB and all working storage mapped in the EVM.

#### **NO**

No ECB is associated with this error. If this option is coded, no working storage will be dumped unless there is a Dump Override Table entry associated with this system error that specifies an area of working storage, or unless a portion of working storage has been specified using the LIST or SLIST parameters.

### **LIST**

Specifies the location of one or more LISTC macro calls, which are used to identify additional main storage areas to be dumped. Specify one of the following:

#### *label2*

A symbolic label.

*Ry* The name of a register containing the address. Valid register names are R1 through R7, R14, and R15. *Ry* cannot specify the same register as *Rx* or *Rz*. If *senum* is coded as a register, SLIST can not specify the same register as *senum* and cannot specify R14 or R0. Parentheses cannot be used. *Ry* cannot be R0 if MSG=YES is coded.

### **SLIST**

Specifies the address of a user-defined list of storage areas to be dumped. Specify one of the following:

#### *label3*

A symbolic label (EBW034, for example).

*Rz* The name of a register containing the address. Valid register names are R0 through R7, R14, and R15. *Rz* cannot specify the same register as *Rx* or *Ry*. *Rz* cannot be R0 if MSG=YES is coded.

### **SYSDUMP**

Specifies whether a dump of system storage will be performed.

#### **YES**

This is the default for control program errors.

**NO**

This is the default for operational program errors. When SYSDUMP=NO, dump overrides created with the IDOTB macro or the ZIDOT command have no effect.

**Note:** System storage will always be dumped if there is no ECB associated with the error.

**DSECT=YES**

This format of the macro is used to generate a DSECT describing the macro expansion, and the equates associated with various macro options. It is intended for CP use only.

**OPR**

Specifies whether the dump will be an OPR or a CTL dump as follows:

**YES**

Specifies that if the dump is in the control program, the dump will display as an OPR dump. This allows the dump to be controlled by the ZASER command with the DUPL or NODUPL parameter. You can specify this value only if you specify NO for the SYSDUMP parameter.

**NO**

Specifies that if the dump is in the control program, the dump will display as a CTL dump.

## Entry Requirements

If an ECB is associated with the error, the address of the ECB must be in R9. This address must be valid for the address space in which the macro is invoked.

## Return Conditions

- Control is returned as specified by action as follows:
  - E** To EXITC processing
  - R** To NSI in the calling program
  - C** To catastrophic processing.
- On return to NSI, the contents of all registers are preserved.

## Programming Considerations

- The positional parameters (MSG, OP, CP, ECB, NEB, NBL) from prior releases of the TPF system are supported only for migration purposes. Use the keyword parameters described previously for any new SERRC macro calls.

**Notes:**

1. If no keyword parameters are used, the old format macro expansion will be generated and no prefix character will be concatenated with the system error number.
  2. The prefix character will default to 'U' for SERRC calls in user programs from prior releases that have not been reassembled.
- If this macro is issued with an exit before 1052 state, state change will be disabled for the failing MDBF subsystem.
  - If this macro is issued with an exit while state change is in progress, and the system error was caused by the BSS state change ECB, the error will be forced

## SERRC

irrecoverable. If the error was caused by the state change ECB for a non-BSS subsystem, the subsystem will be cycled down and state change will be disabled for the subsystem.

- The Selective Memory Dump Table in copy member CPST of the CCCPSE CSECT defines large data areas in main storage that are normally not dumped. If data from one of these areas is required for problem diagnosis, the user must create a dump override for the system error by coding an IDOTB macro call in copy member CUDP of the CCUEXT CSECT. (For IBM system errors, update copy member CIDP in the CCCPSE CSECT.)
- If MSG=YES is specified, a user-supplied message will be sent to CRAS in addition to the system error message which is normally sent by the system error routine. The user must load R0 with the address of the message prior to executing the SERRC macro. The maximum size of this message is 255 bytes including the character count, which must appear in the first byte. Control characters (such as line feed and EOM) are not permitted; however, the carriage return character is supported.

**Note:** The GENMSG macro with the TEXTONLY option may be used to build the correct message format.

- If LIST is specified, the user must code a valid sequence of LISTC macros at the specified location. Refer to the LISTC macro description for more information.
- If SLIST is coded, the user must build a list containing the parameters to be used in dumping additional storage. Each entry in the list consists of 3 words. The first byte of the first word is an indicator byte, and the last 3 bytes of the first word are unused. The second word contains a starting address. The third word contains an ending address or the number of bytes, less one, to be dumped. The format of the list is as follows:

- 1st word, first byte

The list entry is ignored if the value provided is other than those defined.

**X'00'** HEX dump request (3rd word contains an address).

**X'01'** EBCDIC dump request (3rd word contains an address).

**X'02'** HEX dump request (3rd word contains length –1).

**X'03'** EBCDIC dump request (3rd word contains length –1).

**X'F0'** Ignore this entry.

**X'FF'** End of list.

- 2nd word contains the starting address of storage area to be dumped.
- 3rd word contains the ending address of storage area to be dumped, or number of bytes to be dumped, minus one.

### Notes:

1. The list must start on a fullword boundary.
  2. The storage actually dumped will begin and end on a doubleword boundary, and will include all bytes specified, up to a maximum of 4096 bytes.
  3. If the start address is greater than the end address, a doubleword of storage beginning with the start address is dumped.
- If the location specified by the LIST parameter is in memory belonging to another ECB, even in VEQR mode, the dump contains the phrase Address Error for the LISTC with the erroneous address.



## Examples

- The following is an example of an operational program system error:

```

XYZ SERRC E,(CE93C0),PREFIX=A OPR-A0003C0 with exit

 SERRC R,(CE91E4),PREFIX=Q, OPR-Q0001E4 with return, use
 SLIST=EBW008 area at EBW008 defined as
 follows:

EBW008: X'01' EBCDIC dump, 3rd word contains address
EBW012: X'000D3180' Start of area to be dumped
EBW016: X'000D35A0' End of area to be dumped
EBW020: X'02' HEX dump, 3rd word contains length - 1
EBW024: X'000E4400' Start of area to be dumped
EBW028: X'000000FF' Length -1 of area to be dumped
EBW032: X'F0' Ignore this entry
EBW036: ----- Irrelevant
EBW040: ----- Irrelevant
EBW044: X'FF' End of list

```

- The following is an example of an operational program system error:

```

 SERRC E,(CE91F0),PREFIX=H, OPR-H0001F0 with exit and
 LIST=ABC LISTC calls coded as follows:

ABC DS 0F
 LISTC TAG=EBW034, dump EBW034-EBW037
 LEN=4,
 NAME=ITEM1
 LISTC TAG=EBW038, dump 24 bytes at the location
 LEN=24, pointed to by EBW038
 NAME=ITEM2,
 INDIR=YES
 LISTC END terminate the list

```

- The following is an example of a control program system error:

```

 SERRC C,(CECRS235), CTL-I000235 catastrophic
 ECB=NO, no associated ECB
 PREFIX=I IBM system error

```

- The following is an example of a control program system error:

```

 LA R7,CA9S02 CTL-I0000E1 ..
DMPRTN SERRC R,R7,PREFIX=I .. with return

```

- The following is an example of a control program system error:

```

 LA R0,DPMMSG3 message defined below
SERRC E,(CE929C), CTL-I00029C with exit
 MSG=YES, message address in R0
 PREFIX=W IBM system error

DPMMSG3 DC AL1(L'SDP3)
SDP3 DC C'SIPC WRITE ERROR - INDETERMINATE'

```

- Generate the macro expansion DSECT

```

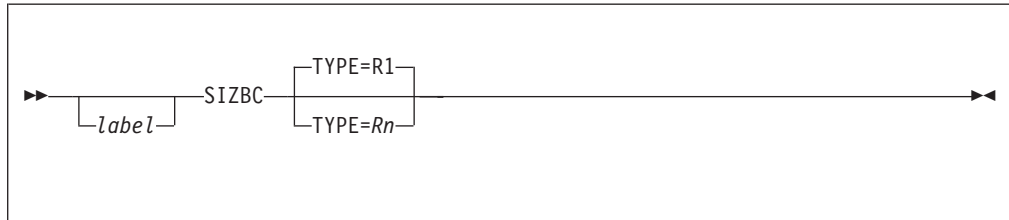
 SERRC DSECT=YES

```

## SIZBC–Obtain Logical Size

This general macro obtains the logical size of a storage block.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**TYPE=Rn|R1**

The register specified in this parameter contains the logical storage block type equate value needed to obtain the logical size of the storage block requested. The logical size of the storage block is returned in the register specified.

This is an optional keyword. If the parameter is omitted, the default assignment is R1. R0 is not used for any calls and R8–R13 are invalid for E-type calls.

### Entry Requirements

R9 must contain the address of the ECB being processed.

### Return Conditions

- Control is returned to the next sequential instruction.
- The register specified in the TYPE statement will contain the logical size of the storage block.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

### Programming Considerations

- R0 may not be used with this macro.
- This macro can be executed on any I-stream.
- A system error dump can occur when servicing a SIZBC request. See *Messages (System Error and Offline)* and *Messages (Online)* for detailed information.

### Examples

```

 LA R5,L1 381 Byte Block type
 SIZBC TYPE=R5 Find logical size of L1 type block
* R5 will now contain hex 17D (decimal 381)

```

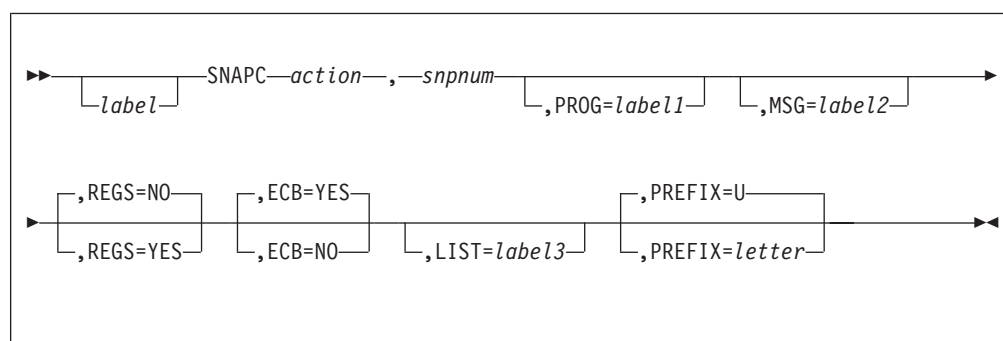
## SNAPC—Snapshot Dump

This general macro, similar to the system error macro (SERRC), collects diagnostic data to be used for problem determination. If the amount of data needed to analyze a problem is limited and the location of the data can be predetermined, a snapshot dump can be used instead of the typical SERRC dump to reduce the use of system resources and make the diagnostic data more precise.

The storage dump locations are specified by LISTC macros indicated in the LIST parameter.

The SNAPC macro may be used whenever an operational program (E-type) or the control program (C-type) detects an error.

## Format



### *label*

A symbolic name may be assigned to the macro statement.

### *action*

Required. This positional parameter specifies the action to be taken by the SNAPC service routine upon completion of the dump:

- R** Return to next sequential instruction (NSI)
- E** Exit the entry control block (ECB) If E is coded for this parameter the ECB exits irrespective of the specification of ECB parameter.

### *snpnum*

Required - this positional parameter specifies the snapshot dump code. There are three ways to code this information:

#### *(errcode)*

A symbolic equate for a snapshot error code, enclosed in parentheses.

*sd* A hexadecimal system error number in the range X'0-7FFFFFFF'.

*Rx* The symbolic name of a register which contains the snapshot error code. Valid symbols are R0 through R7, R14, and R15.

The defined prefix is appended to the front of the error number.

### **PROG=*label1***

Optional - this parameter specifies the location of the program name to be used in the snapshot dump. When specified, the PROG keyword is the label of an addressable character constant field. The length of the constant is the length of the program name (maximum length is 16 bytes). If not specified, the four-byte

## SNAPC

program name located four bytes off R8 is used in the snapshot dump and the console message. This parameter should be coded when R8 does not point to a valid program.

### MSG=*label*2

Optional - this parameter specifies the address of the appended message. When specified, the MSG operand is the label of the message (MESSAG) of the form :

```
MESSAG EQU *
MSGLEN DC AL1(L'MSGSTUF)
MSGSTUF DC C'THIS IS THE APPEND MESSAGE'
```

### REGS=NO|YES

Optional - this parameter specifies whether the general registers are to be included when a snapshot dump is displayed to the system console. **The default value is NO.**

**Note:** The registers will always be included in the snapshot dump when formatted by either ICDF or STPP.

### ECB=YES|NO

Optional - this parameter specifies whether the SS, SSU, and terminal address are to be taken from the ECB. **The default value is YES.**

If ECB=NO:

- The SS and SSU are forced to the BSS and
- The terminal address is unavailable and will be displayed as N.A.
- Return is to the CPU loop.

ECB=NO should only be coded when no ECB can be associated with the program in error. ECB=NO can only be coded when running in the SVM.

### LIST=*label*3

Optional. This parameter specifies the location and length of the data to be dumped. If not specified, no data area will be dumped. When specified, this LIST parameter is the label of the data areas specified by LISTC macros.

### PREFIX=U|*letter*

Optional. This parameter specifies the prefix of the snapshot dump code. By using a prefix an application will have a SNAPC code name space associated with it. When coded, the PREFIX keyword is a single alphabetic character. **The default prefix is U. The letters I and W-Z are reserved for IBM use.**

## Entry Requirements

- If an ECB is associated with the error, R9 must contain its address.
- If PROG parameter is not coded, R8 is assumed to point to a standard E-type program header.

## Return Conditions

- Control is returned, as specified by the first positional parameter *action*, to NSI when 'R' is indicated or to the EXITC macro processing when 'E' is indicated.
- A storage dump is taken and the system operator is informed of the error via a CRAS set if the console notification is not suppressed by the ZASER command with the SNAP and CONSOFF parameters specified.
- The condition code and the contents of all registers are preserved on return to NSI.

## Programming Considerations

- The SNAPC service routine will use up to eight 4KB blocks to collect the specified data. Overflow data will be truncated.
- If a page fault occurs while SNAPC is processing a requested dump in VEQR mode (because an address belongs to another ECB), SNAPC continues processing the requested dump. If a page fault occurs in an application running in VEQR mode, a SNAP dump is generated to log the application's address violation for correction.

## Examples

This example forces a snapshot dump bearing ID number 12345 and a prefix of 'A' to be issued. Control will return to the program after the dump and the registers will be included in the dump. The ECB will be used for the SS and SSU names, and the terminal ID and the program name will at label NAME. The LISTC definitions are at label SNAPSTUF and the appended message will be at label MESS.

```

 SNAPC R,12345,PROG=NAME,MSG=MESS,REGS=YES,ECB=YES, *
 LIST=SNAPSTUF,PREFIX=A
:
MESS DC AL1(L'MSGSTUF)
MSGSTUF DC C'PROGRAM BLEW UP'
NAME DC C'C001'
SNAPSTUF LISTC NAME=MYSTUFF,TAG=EBW000,LEN=4,INDIR=YES
 LISTC NAME=DATA10,TAG=CE1CR0,LEN=1000,INDIR=YES
 LISTC END

```

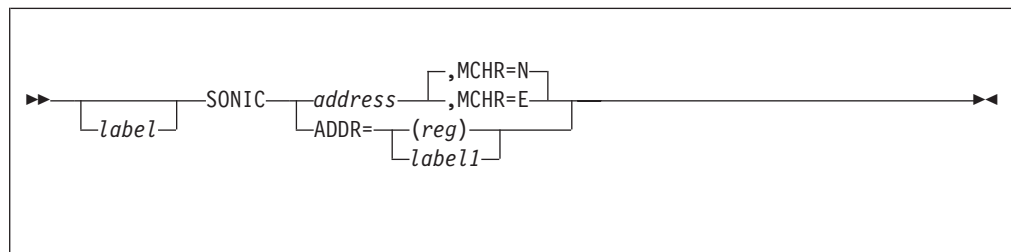
## SONIC—Obtain Symbolic File Address Information

This general macro performs two functions. Given a symbolic file address, this macro will return the characteristics of that file address. Characteristics include:

- Addressing mode and record type: FARF3, FARF4, FARF5 (fixed or pool), or FARF6.
- Record size: small, large or 4K.
- Record duplication: duplicated or nonduplicated.
- For a pool file address reference format (FARF) address, the duration: short-term or long-term.
- Record scope: whether a fixed FARF address is common to all processors, I-streams and subsystem users, or whether it is unique to a subset of all processors, I-streams, and subsystem users.

The second function of this macro is to validate the input MCHR.

### Format



*label*

A symbolic name can be assigned to the macro statement.

*address*

Location of the file address.

*fileaddress*

A file address reference word (D0–DF) which contains the file address.

*(reg)*

A register (R0–R7) which contains the file address.

#### MCHR

This parameter is used when the input file address is an MCHR address. When an MCHR address is provided, the first parameter must be a data level index.

**N** Default. File address is not an MCHR address

**E** File address is a 7-byte MCHR address with the module number contained in CE1FMx and the BBCCHHR contained in CE1FXx of the specified level.

**ADDR=(reg)|label1**

The general register (R1–R7) containing the location, or a label indicating the location, of an 8-byte file address.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The symbolic file address must be contained in file address reference word (FARW) CE1FMx, where x is the specified ECB data level; or the symbolic file address must be in a register or, if it is an 8-byte file address, it must be pointed to by the ADDR parameter.

- If the MCHR=E option is used, the symbolic module number MM must be the first 2 bytes in CE1FMx and the 5-byte BBCCHHR address must be in CE1FXx, offset by 2. For example, assuming the MCHR address is in EBX000:

```
MVC CE1FM1(2),EBX000
MVC CE1FX1+2(5),EBX000+2
```

would properly set up the MCHR for the E parameter.

## 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 ECB data level is unchanged if an ECB data level is specified. The location pointed to by the ADDR parameter is unchanged if ADDR is specified.
- The contents of R0–R7 remain the same as on input. If one of these registers is used to input the file address, it will contain the file address characteristics on return. These characteristics have equates associated with them.
- R14 will point past the macro decoder linkage generated by the macro expansion.
- The input parameter register, or R15 if an ECB data level or ADDR is used, will contain the following bit settings based on the symbolic file address characteristics:

| Symbol               | Bits  | Description:                                                                                                                                                                             |
|----------------------|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SONIC_TYPE_INDICATOR | 00    | Type indicator<br>0       fixed<br>1       pool                                                                                                                                          |
| SONIC_POOL_LONGEVITY | 01    | Pool longevity indicator (refer to 1)<br>0       long-term pool<br>1       short-term pool                                                                                               |
| SONIC_ADDRESS_FORMAT | 02–03 | Address format indicator<br>00       FARF3 addressing format<br>01       FARF4 addressing format<br>11       FARF5 addressing format<br>10       FARF6 addressing format<br>04–06   Zero |
| SONIC_FADDR_INVALID  | 07    | File Address Validity indicator<br>0       Valid File Address<br>1       Invalid File Address<br>08–15   Zero                                                                            |

## SONIC

| Symbol              | Bits | Description:                                                                                                                                                                                                                                     |
|---------------------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SONIC_RECORD_UNIQUE | 16   | Record ownership indicator (refer to 2)<br><br>0 Address is common to all SSUs/processors/I-Streams of the caller's subsystem<br><br>1 Address is unique to some combination of SSUs/processors/I-Streams within the subsystem.                  |
| SONIC_ALT_REC_SIZE  | 17   | Alternate record size indicator<br><br>0 size is indicated by bit 31 alone<br><br>1 size is 4K (4096) bytes. Bit 31 should also = 1<br><br>18–27 Zero<br><br>28 Pool address indicator<br><br>0 fixed address<br><br>1 pool address (refer to 3) |
| SONIC_DUPLICATED    | 29   | Duplication indicator<br><br>0 not duplicated/no fallback<br><br>1 duplicated/fallback<br><br>30 Always set to 1                                                                                                                                 |
| SONIC_REC_SIZE      | 31   | Record size indicator<br><br>0 small size record (381 bytes)<br><br>1 large size record (1055 or 4096 bytes)                                                                                                                                     |

### Notes:

1. This bit is meaningful only when the file address is a pool address (bit 0 = 1) and will be zero otherwise.
2. This bit is meaningful only when the file address is a fixed address.
3. This bit has no equate associated with it. It is always set to 1 to maintain compatibility with an earlier interface.
4. For a SONIC using the MCHR option, R15 will be set to an index specifying the failing element of the input MCHR address.

| Equate name        | Bit | Description                   |
|--------------------|-----|-------------------------------|
| SONIC_RTN_GOOD     | 0   | Valid MCHR address            |
| SONIC_RTN_INV_CYL  | 4   | Invalid cylinder value        |
| SONIC_RTN_INV_HEAD | 8   | Invalid head value            |
| SONIC_RTN_INV_MOD  | 12  | Invalid or not mounted module |
| SONIC_RTN_INV_BIN  | 16  | Invalid bin value.            |

The record number is not validated. There is no way to validate the record number except by attempting to read the actual record because the size of the record cannot be determined from its address.



## Programming Considerations

This macro can be run on any I-stream.

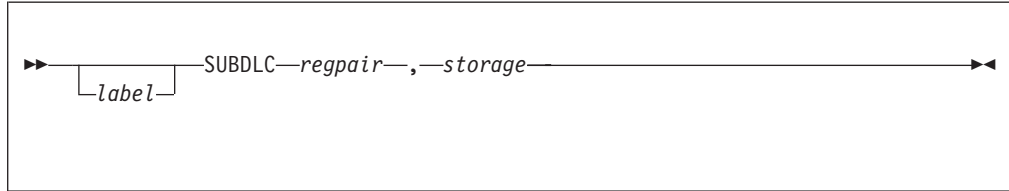
## Examples

None.

## SUBDLC—Subtract Doubleword Unsigned

Use this general macro to subtract two unsigned doublewords.

### Format



*label*

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

*regpair*

is the even register of an even-odd pair of registers that contain the first operand (a doubleword value) and will contain the result of the arithmetic subtraction.

*storage*

is the label of a storage location that contains the second operand (a doubleword value) that will be subtracted from the first operand in *regpair*.

### Entry Requirements

None.

### Return Conditions

The result of the operation will be in *regpair*. The condition code will be set as if an SL assembler instruction was used.

### Programming Considerations

None.

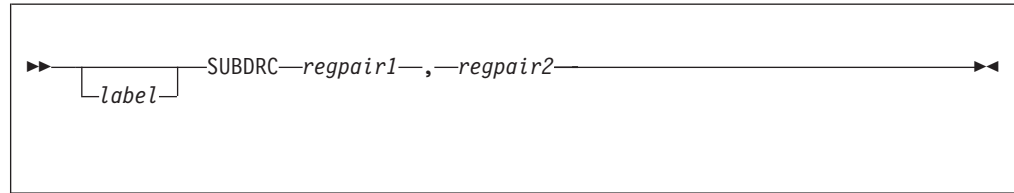
### Examples

None.

## SUBDRC—Subtract Doubleword Unsigned

Use this general macro to subtract two unsigned doublewords.

### Format



*label*

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

*regpair1*

is the even register of an even-odd pair of registers that contain the first operand (a doubleword value) and will contain the result of the arithmetic subtraction.

*regpair2*

is the even register of an even-odd pair of registers that contain the second operand (a doubleword value) that will be subtracted from the first operand in *regpair1*.

### Entry Requirements

None.

### Return Conditions

The result of the operation will be contained in *regpair1*. The condition code will be set as if an SLR assembler instruction was used.

### Programming Considerations

None.

### Examples

None.

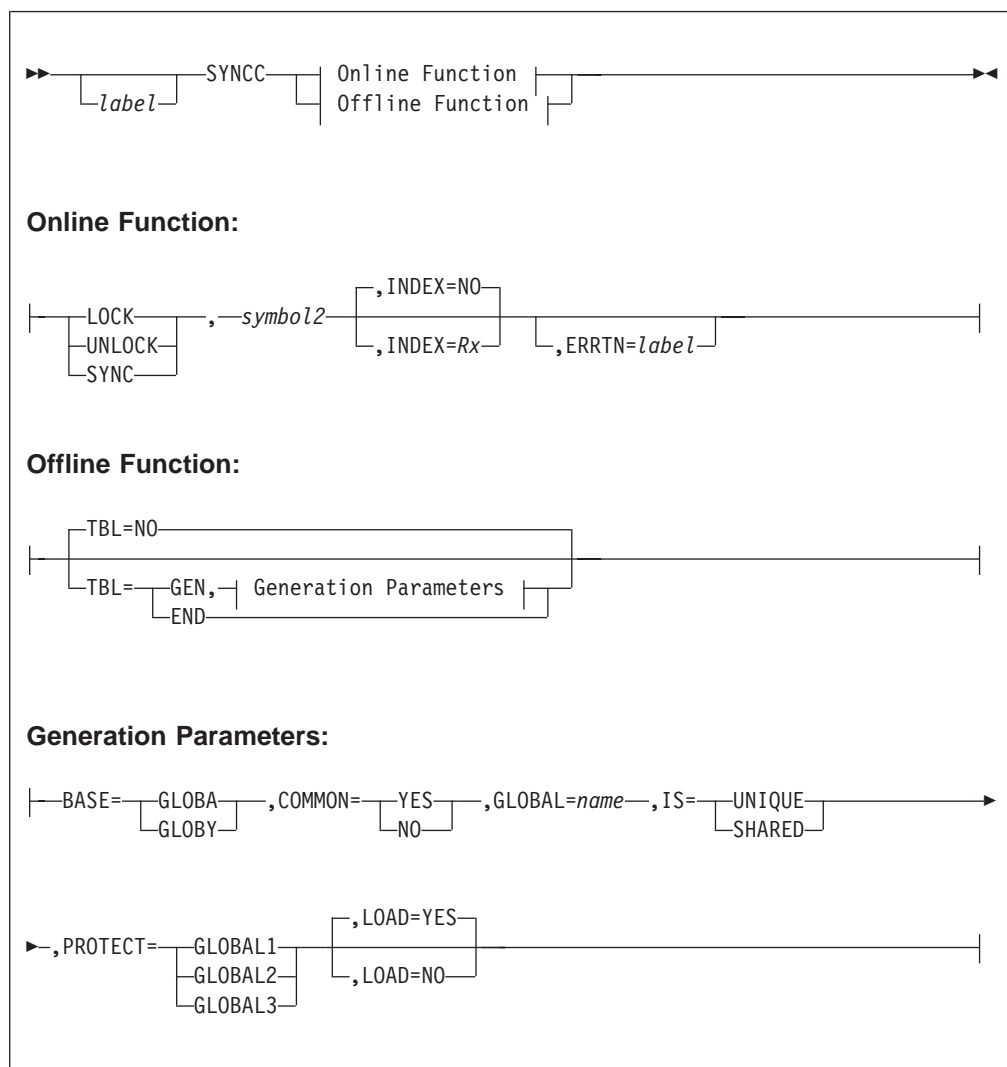
## SYNCC—Synchronize Globals

This general macro has two distinct functions:

1. In the online environment, it permits the initiation of a request to synchronize a global field or record among all active tightly coupled processors, loosely coupled processors, or both.
2. In the offline environment, it is used in SIP Stage 2 to build the system interprocessor global table (SIGT).

See also the global area program material in the *TPF System Installation Support Reference*.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### LOCK

Obtains exclusive use of the specified global. When the LOCK option is used, this macro will obtain the requested global with Hold and will refresh the core copy of the global with the current file copy.

**UNLOCK**

Releases exclusive use of the specified global. The UNLOCK option will unhold the file copy of the global. The requested global must have been locked first.

**SYNC**

Requests synchronization of the specified global. When the SYNC option is requested, the designated global is filed, and an entry is created to activate the synchronization of the global among the active I-streams. In addition, in a loosely coupled (LC) environment, the interprocessor communication facility (IPC) is activated. The IPC is used to inform the other processors in the LC complex that the requested global value should be refreshed in core. After filing the record and activating the IPC, processing is the same as for the UNLOCK option. Again, the requested global must be locked prior to executing this request.

*symbol2*

A global field or record name must be specified as *symbol2*.

**INDEX=NO|Rx**

When *Rx* is specified, the index value in *Rx* is added to the value generated by *symbol2* to calculate the index number for the global record to be synchronized. If this option is not used, the index value generated by *symbol2* is used.

This option is valid only for synchronizable global records (not global *fields*). Valid registers are R0-R7, R14, and R15.

**Note:** The purpose of providing the INDEX feature is to facilitate stepping through a consecutive list of global records in the SIGT. Normal care must be exercised in setting up and using such a list.

**ERRTN=*label***

When present, this parameter will generate a branch to the specified label when an error occurs during execution of the SYNCC macro.

If not present and an error occurs during execution of the SYNCC macro, a system error with dump will occur and the ECB will be exited.

This parameter generates an additional four bytes of object code.

**TBL=NO|GEN|END**

This parameter is specified as GEN or END only when a new system interprocessor global table (SIGT) is to be generated. Since a SIGT is not usually being generated, NO is the default.

- When used, it should be the **only** parameter coded.
- **Application programs should never use this parameter.**

Normal usage is as follows:

```
SYNCC TBL=GEN
SYNCC GLOBAL=...etc.
SYNCC GLOBAL=...etc.
```

```
·
·
·
```

```
SYNCC TBL=END
```

Only the following parameters can be used in the SIGT generation (when TBL=GEN).

```
BASE=
COMMON=
```

## SYNCC

GLOBAL=  
IS=  
LOAD=  
PROTECT=

### **BASE=GLOBAL|GLOBY**

Global base for this data.

### **COMMON=YES|NO**

This indicates whether the global is common to other SSUs or not. YES and NO are the only valid values. If YES is specified, and GLOBAL refers to a Global *field*, then BASE must be GLOBY.

### **GLOBAL=name**

Name of the synchronizable global data.

### **IS=UNIQUE|SHARED**

This indicates whether the global is shared by all I-streams in each central processing complex (CPC). UNIQUE and SHARED are the only valid values.

### **LOAD=YES|NO**

This is valid for global *fields* only. If YES, indicates that on the load of a new SIGT, the DASD-resident copy of the Synchronizable Global Field Record (SGFR) for the item referenced by GLOBAL is to be initialized from the core copy of the global field. If NO, the SGFR record will not be initialized during IPL. YES and NO are the only valid values.

The default is YES.

### **PROTECT=GLOBAL1|GLOBAL2|GLOBAL3**

This indicates which protection key should be stored in the SIGT table entry for this item.

## Entry Requirements

- The following are entry requirements for the online function:
  - R9 must contain the address of the ECB being processed.
  - The entry must not be holding a lock on any global field or global record at the time of a LOCK request.
  - The entry must not be holding a record at the time a LOCK request is made.
  - There should not be any outstanding I/O at the time this macro is issued.
  - The entry must be holding a lock on the global field or global record requested to be unlocked or synchronized.
  - If the parameter INDEX=*rx* is coded, the general register specified by *rx* must contain the increment that will be added to the index generated from *symbol2* to calculate the total global index.
  - If the user wants control returned on an error condition, the ERRTN parameter must be coded.
  - The user should issue the LOCK and associated UNLOCK in the same segment, to ensure the protect key. The LOCK should not be carried across Enter macros.
- The following are entry requirements for the offline function:
  - The macro SYNLST, generated as part of the system generation procedure, must be available to the assembler that processes the SYNCC table generation run.
  - The macro SYNLST contains a list of valid synchronizable global fields and records as defined at system generation time by means of GLSYNC

statements. The sequence of SYNCC table generation statements must exactly follow the order of the SYNLIST sequence, with no omissions or additions.

## Return Conditions

- The following are return conditions for the online function:
  - The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
  - LOCK option:
    - After the lock operation has successfully completed, control is returned to the next sequential instruction (NSI). If the ERRTN parameter is coded, return is NSI+4.
    - The protect key corresponding to the requested global is set in the current PSW.
    - If an error has occurred and the ERRTN parameter is not coded, the entry is exited after a system dump is taken. If the ERRTN parameter is coded, control is returned at the NSI, except for errors stemming from possible destruction of core. In this case, the entry is exited after a system error.
    - If an error is returned from the internally-generated FIWHC macro while trying to retrieve the global sync records, one of the following occurs:
      - If the error is the result of a hold lockout or processing of the ZECBL command (CE1SUD=X'81'), it is a user error. The 0000DB system error is issued and the ECB is exited.
      - If the error occurred because of record retrieval problems (for example, a record ID failure occurred), it is related to database corruption. The 0006A3 system error is issued and the affected subsystem is cycled to 1052 state so that you can take immediate corrective action.
    - The contents of the general registers, except for R14 and R15, and the condition code are saved while processing this macro.
    - The core copy of the requested global has been refreshed by the lock operation.
    - The ECB global sync indicator (CE1SYN) indicates a LOCK condition and the entry's hold counter is incremented.
  - UNLOCK option
    - Control is returned to the next sequential instruction. NSI+4 if ERRTN is coded.
    - The current PSW protect key is reset to that of working storage.
    - The contents of the general registers, except for R14 and R15, and the condition code are saved during execution of this macro.
    - The ECB global sync indicator (CE1SYN) is cleared and the entry's hold counter is decremented.
  - SYNC option
    - Control is returned to the next sequential instruction. NSI+4 if ERRTN is coded.
    - The current PSW protect key is reset to that of working storage.
    - The contents of the general registers, except for R14 and R15, and the condition code are saved during execution of this macro.
- For the offline function, assuming no errors, assembling a valid set of SYNCC statements will produce a SIGT table in the form of a loadable object deck plus a table listing. Since no executable code is generated, there are no return conditions.

## SYNCC

### Programming Considerations

For the online function, this macro can be executed on any I-stream.

### Examples

- This example shows a global symbol @SYNCD being locked, some processing taking place, and then being unlocked.

```
SYNCC LOCK,@SYNCD
```

```
 .
 <processing>
```

```
 .
SYNCC UNLOCK,@SYNCD
```

- The global referred to by @SYNCREC plus the additional index value in R4 is filed and synchronized among the active i-streams. If an error should occur, control transfers to LABEL99 for recovery.

```
SYNCC SYNC,@SYNCREC,INDEX=R4,ERRTN=LABEL99
```

- This is an example of generating a system interprocessor global table. First, the TBL parameter is used to initiate table generation. Next, a number of globals and their characteristics are defined. Finally, table generation is ended with the TBL parameter again. The global shown below is being defined in global area A with the protection key of global1. The global is unique to the SSU defining it, is I-stream unique, and when the SIGT is loaded the global value in main memory is synchronized with its DASD copy.

```
SYNCC TBL=GEN
```

```
SYNCC GLOBAL=@SYBCD,BASE=@GLOBA,PROTECT=GLOBAL1,COMMON=NO,LOAD=YES, X
 IS=UNIQUE
```

```
SYNCC etc. .
```

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

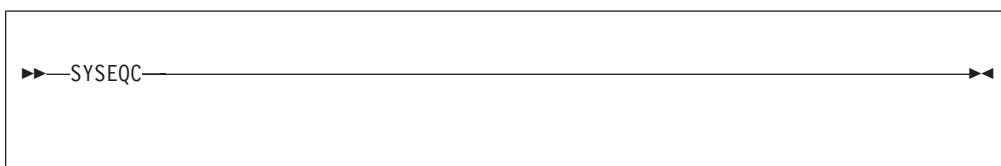
```
SYNCC TBL=END
```



## SYSEQC—Configuration-Dependent System Equates

This general macro contains record type equate names and values created by the FACE Table Generator.

### Format



### Entry Requirements

None.

### Return Conditions

Control is returned to the next sequential instruction.

### Programming Considerations

- This macro can be executed on any I-stream.
- This macro is created by the FACE Table Generator.
- Interfaces with the FACE Table. During the assembly process, SYSEQC declares and assigns a value to a record type name. The value assigned to this record type name is used as an index into the FACE table (FCTB).
- SYSEQC equates are often used by FACE calls. When SYSEQC equates change, segments which have FACE calls that use the equates which have changed, must be reassembled. To avoid reassembling code, FACS should be used instead of FACE.
- All defined symbols in SYSEQC must begin with the pound sign character (#).
- If the FACE table changes, SYSEQC must be updated to reflect the change for the specific subsystem.
- The last equate defined for the FACE values in SYSEQC must be for #FACTEND.

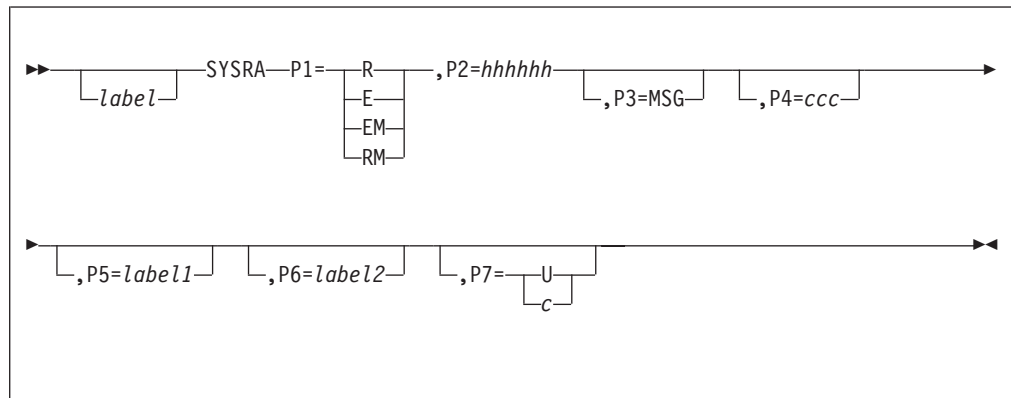
### Examples

None.

## SYSRA—Application System Error

This general macro is used by application programs to initiate error processing. It is used to communicate basic error information to the computer operator via the control program SERRC macro, and to start appropriate user recovery programs.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **P1**

This is required. It specifies where control is to be given after processing.

**E** EXITC after SERRC.

**R** Return to NSI after SERRC.

#### **EM**

EXITC after calling the Error Monitor.

#### **RM**

Return to NSI after calling the Error Monitor.

#### **P2=hhhhh**

This is required. It must be a hexadecimal error number in the range X'1-FFFFFF'.

#### **P3=MSG**

This is a message option.

If coded, it must be MSG. It specifies that R0 contains the address of additional message text destined for the CRAS. The first byte at this address is the character count. The message can be a maximum of 255 bytes.

#### **P4=ccc**

These are the last three characters of the data record fix program to be executed after the Error Monitor. It is optional. If coded, P1=EM or P1=RM must be coded.

If coded, it must be one of the following: XMT, XEX, XFC, XID, XIG, XIM, XNC, XNA, XNI, XNF, XNS, XRT, XO, XOR, XPI, XPD, XPG, XPW, XPQ, XTF.

#### **P5=label1**

This is the symbolic location to which control will be given if a hardware error has been detected by the CP and either P1=R or P1=RM has been coded. It is optional.

**P6=***label*2

The symbolic location where control will be transferred if an invalid file address has been detected by the CP and either P1=R or P1=RM has been coded. The parameter is optional.

**P7**

The prefix character that will be concatenated with the system error number if P1=R or P1=E has been coded and a SERRC is issued. The letters I and W through Z are reserved for IBM use. The default is P7=U.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- If P1 = E is specified, an EXITC will be generated.
- If P1 = R, a return to next sequential instruction (NSI) in the user program will be made.
- If P1 = EM, linkage will be made to the error monitor.
  - If P4 is not specified, an EXITC will be issued from the monitor.
  - If a fix program is specified in P4, either an EXITC will be issued from the Monitor or an ENTDC macro will be issued from SYSRA, effectively exiting from the user program.
- If P1 = RM is specified, linkage to the error monitor and a return to the NSI in the user program is made.
- The condition code may be changed.

## Programming Considerations

- The system error summary byte, CE1SUG, is interrogated for the presence of a hardware error and for an indication of an invalid file address, or both.
- Bit 7 of error byte EBER01 is interrogated to determine if a data record fix program is in progress.
- If P3 = MSG is specified, R0 is utilized and saved across the macro.
- If P1 = RM and P4 is specified, a program level will be used upon entry to a fix program.
- Every agent-initiated entry must receive a response. If the macro SYSRA is used and P1 = R or RM (a return option), the user has the responsibility of sending said response to the agent. If SYSRA is used and P1 = E or EM (an exit option), the user must be aware of the following (assuming we have an agent initiated entry):

## SYSRA

|                      | P1 = E                                                             | P1 = EM and P4<br>not specified                                              | P1 = EM and<br>P4 specified                                                  |
|----------------------|--------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Hardware             | No SERRC macro generated; EXITC no response *.                     | No SERRC generated; agent receives response from control program.            | No SERRC generated; agent receives response from control program.            |
| ID/RCC               | SERRC E, error number generated in SYSRA; agent receives response. | SERRC E, error number generated by control program; agent receives response. | SERRC E, error number generated by control program; agent receives response. |
| Invalid File Address | Same as ID/RCC above.                                              | Same as ID/RCC above.                                                        | Same as ID/RCC above.                                                        |

The option marked with an asterisk (\*) is limited to those users that are internal-initiated rather than agent initiated.

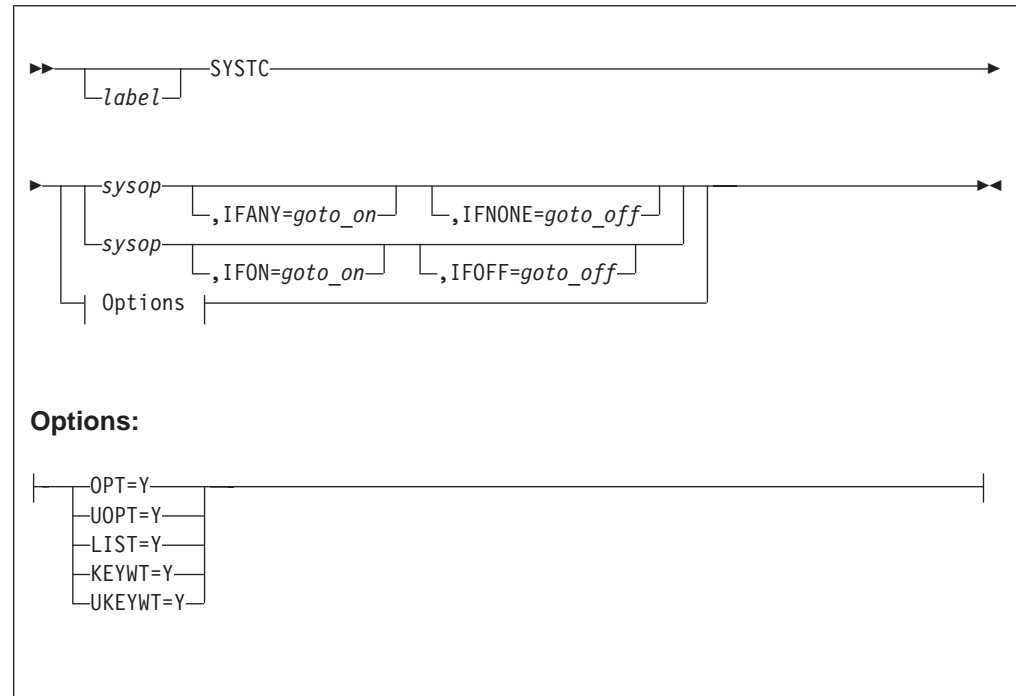
## Examples

None.

## SYSTC–Test Sysgen Options

This general macro is used to determine which system options were included in this installation at system generation. System options are defined in the SYSTG macro and user options are defined in the SYSUG macro. They are maintained as a string of bit switches in the control program. Subsystem-dependent SYSET values must be accessed with the CONKC table. (Refer to OPT parameter below.) The *sysop* positional parameter of SYSTC is mutually exclusive with the OPT, UOPT, LIST, KEYWT, and UKEYWT parameters.

### Format



*label*

A symbolic name may be assigned to the macro statement.

*sysop*<sub>1,2</sub>

The symbolic name for the system option in question. The symbolic name is the SYSET global variable without the preceding ampersand (&). The *sysop* positional parameter is mutually exclusive with the OPT, UOPT, LIST, KEYWT, and UKEYWT options.

**IFANY=goto\_on**

If the system option bit specified by *sysop*<sub>1</sub> is on, then branch to the *goto\_on* label, which can be any valid label in the program segment.

**IFNONE=goto\_off**

If the system option bit specified by *sysop*<sub>1</sub> is off, then branch to the *goto\_off* label, which can be any valid label in the program segment.

**IFON=goto\_on**

If the subsystem option bit specified by *sysop*<sub>2</sub> is on, then branch to the *goto\_on* label, which can be any valid label in the program segment.

## SYSTC

### **IFOFF=goto\_off**

If the subsystem option bit specified by *sysop*<sub>2</sub> is off, then branch to the *goto\_off* label, which can be any valid label in the program segment.

### **OPT=Y**

Defines IBM's set of system option bytes when called from the CP, copy segment CAPT. The OPT parameter is mutually exclusive with all other parameters.

### **UOPT=Y**

Defines the user's set of system option bytes when called from the CP, copy segment CAPT, and the user's set of subsystem option bytes when called from real-time segment CVNT. The UOPT parameter is mutually exclusive with all other parameters.

### **LIST=Y**

Generates a set of equates used by the ZSYSG command handler. The LIST parameter is mutually exclusive with all other parameters.

### **KEYWT=Y**

IBM's section of the ZSYSG keyword table is generated. The KEYWT parameter is mutually exclusive with all other parameters.

### **UKEYWT=Y**

The user's section of the ZSYSG keyword table is generated. The UKEYWT parameter is mutually exclusive with all other parameters.

## Entry Requirements

When a subsystem option is requested, SYSTC will call the CINFC macro to access the CONKC table. Registers R14 and R15 are used for this and must be available for use.

## Return Conditions

When a *sysop* is coded:

- The condition code is set.
- The requested branch is taken as indicated by the new condition code. 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.

When **OPT=Y** is coded:

- IBM's system option bit string is initialized in main storage within CAPT, and IBM's subsystem option bit string is initialized within CVNT.

When **UOPT=Y** is coded:

- The user's system option bit string is initialized in main storage in CAPT, and the user's subsystem option bit string is initialized in CVNT.

When **LIST=Y** is coded:

- The equates used by the ZSYSG command processor are generated.

When **KEYWT=Y** is coded:

- IBM's section of the ZSYSG keyword table is generated in program CSY2.

When **UKEYWT=Y** is coded:

- The user's section of the ZSYSG keyword table is generated within program CSY2.

## Programming Considerations

- This macro, using the *sysop* parameter, can be executed on any I-stream.
- This macro is created by SIP.
- If a subsystem-dependent option bit is specified, this macro requires 16 bytes of storage if one keyword parameter is coded and 20 bytes if both keyword parameters are coded.
- If a system option bit is specified, this macro requires 8 bytes if one parameter is coded and 12 bytes if two are coded.

The following options for coding this macro are restricted to specific programs requiring particular output.

- If OPT=Y is coded, this macro requires 1 byte of storage for each 8 system options. The system option bit switches will be generated by specifying the SYSTC macro with the OPT=Y parameter in program CAPT. The subsystem-dependent options bit switches will be generated by specifying SYSTC with the OPT=Y parameter in program CVNT.
- If UOPT=Y is coded, this macro requires 1 byte of storage for each 8 system options. The system option bit switches will be generated by specifying the SYSTC macro with the UOPT=Y parameter in program CAPT. The subsystem-dependent options bit switches will be generated by specifying SYSTC with the UOPT=Y parameter in program CVNT.
- If LIST=Y is coded, this macro generates equates, but it does not generate any storage. The equates are generated by specifying the SYSTC macro with the LIST=Y parameter in programs CSY0 and CSY1.
- If KEYWT=Y is coded, the amount of storage generated depends on the number of IBM SYSTC bytes defined in the SYSTG macro. The number of bytes generated is 64 times the number of IBM SYSTC bytes defined. The table is generated by specifying the SYSTC macro with the KEYWT=Y parameter in program CSY2.
- If UKEYWT=Y is coded, the amount of storage generated depends on the number of user SYSTC bytes defined (see the SYSUG macro code). The number of bytes generated is 64 times the number of user SYSTC bytes defined. The table is generated by specifying the SYSTC macro with the UKEYWT=Y parameter in program CSY2.

## Examples

- Macro Calls with *sysop*  
 (symbol, set condition code and return to the next sequential instruction)  
 NAME1 SYSTC SBCONS  
 (symbol, subsystem-dependent option bit)  
 NAME2 SYSTC SBAMSRC,IFOFF=NAME2  
 (symbol, system option bit)  
 NAME3 SYSTC SBVFA,IFANY=NAME4  
 (symbol, request system copy of subsystem-dependent option)  
 NAME4 SYSTC SBOLDGF,IFON=NAME6,IFOFF=NAME7
- Macro Calls encoded in CVNT and in CAPT  
 NAME5 SYSTC OPT=Y  
 NAME6 SYSTC UOPT=Y
- Macro Call encoded in programs CSY0 and CSY1  
 NAME7 SYSTC LIST=Y

## SYSTC

- Macro Calls encoded in program CSY2

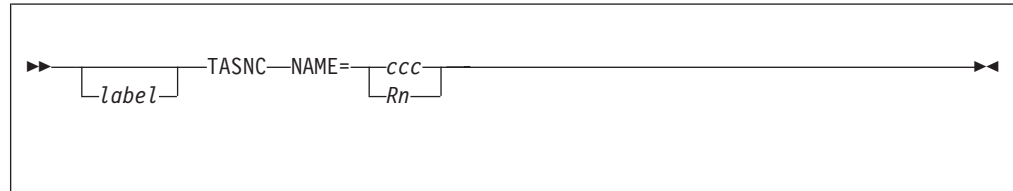
```
NAME8 SYSTC KEYWT=Y
NAME9 SYSTC UKEYWT=Y
```



## TASNC—Assign General Tape

This macro assigns the specified general tape to the ECB issuing the TASNC macro.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **NAME**

Specifies the symbolic general tape name. It can be:

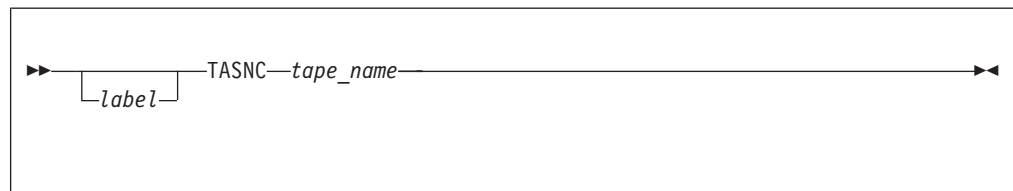
#### *ccc*

A 3-character string representing a symbolic general tape name. The first 2 characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first 2 characters cannot be RT.

#### **(R*n*)**

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.

The following macro format is still supported:



#### *label*

A symbolic name can be assigned to the macro statement.

#### *tape\_name*

A 3-character symbolic general tape name must be specified as the first positional parameter.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must be open when this macro is issued.

## 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 tape position is unchanged.

## TASNC

### Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
- If the specified tape was not reserved before this macro was issued, processing of the entry is suspended until the tape is placed in reserved status. This implied wait can cause the entry to give up control of the system.
- If this macro is issued prior to the end of tape restart, the ECB is exited and a system error issued.

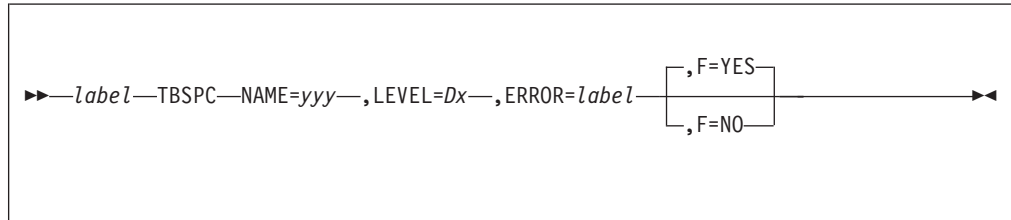
### Examples

None.

## TBSPC—Backspace General Tape and Wait

This macro moves the specified general tape backward over a specified number of physical blocks.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic general tape name. It can be:

*yyy*

A three-character string representing a symbolic general tape name. The first two characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first two characters cannot be RT.

**(R*n*)**

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=D*x***

A symbolic data level (D0–DF) must be specified.

**ERROR=*label***

The label of an operational program error routine within the current program segment must be specified.

**F** An optional keyword parameter can be specified indicating whether fallback to a previous volume is to be allowed when processing multivolume files. This will occur when an attempt is made to backspace a tape past the load point.

**YES**

Fallback is to be allowed.

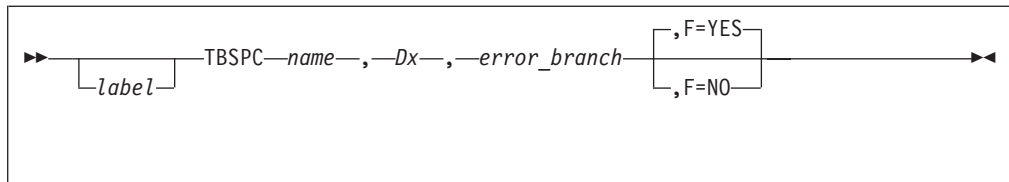
**NO**

Fallback is to be inhibited.

If omitted, a default of YES is assumed.

The following macro format is still supported:

## TBSPC



### *label*

A symbolic name can be assigned to the macro statement.

### *name*

A 3-character symbolic general tape name must be specified as the first parameter.

*Dx* A symbolic data level (D0–DF) must be specified as the second parameter.

### *error\_branch*

The label of an operational program error routine within the current program segment must be specified as the third parameter.

**F** An optional keyword parameter can be specified indicating whether fallback to a previous volume is to be allowed when processing multivolume files. This will occur when an attempt is made to backspace a tape past the load point.

### **YES**

Fallback is to be allowed.

### **NO**

Fallback is to be inhibited.

If omitted, a default of YES is assumed.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must be open when this macro is issued.
- The number of physical blocks to be backspaced must be contained in the low-order 2 bytes of the file address reference word (FARW) of the data level specified by this macro. This number cannot be zero and cannot exceed 65 535. Attempts to backspace more than 65 535 records will backspace the number of records equal to the value entered mod 64 K.

## Return Conditions

- If no I/O hardware errors or unusual conditions have occurred, control is returned to the next sequential instruction (NSI). Otherwise, control is returned to the error routine.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- All pending I/O operations (including this request) are complete for this ECB.
- For I/O hardware errors, the system error routine has taken a storage dump and informed CRAS.

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**

- This macro cannot be used to backspace a blocked tape. If backspacing is attempted with a blocked tape, a system error is issued.

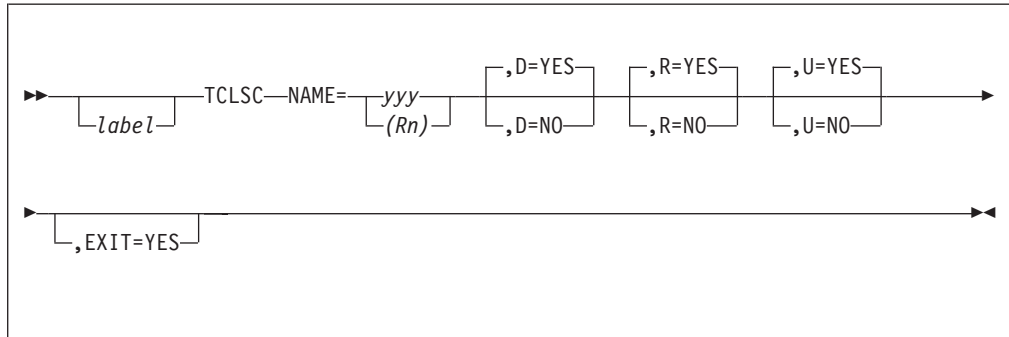
## **Examples**

None.

## TCLSC—Close a General Tape

This macro returns a specified general tape to the system.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **NAME**

Specifies the symbolic general tape name. It can be:

##### *yyy*

A three-character string representing a symbolic general tape name. The first two characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, 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.

#### **D=YES|NO**

An optional parameter can be coded to indicate if a dismount is to be performed for this tape, which would remove the tape name from the tape status table. If specified, this parameter must be coded as YES or NO. If omitted, a default of YES is assumed.

#### **R=YES|NO**

An optional parameter can be coded to indicate whether a rewind is to be performed for this tape. If specified, this parameter must be coded as YES or NO. If omitted, a default of YES is assumed.

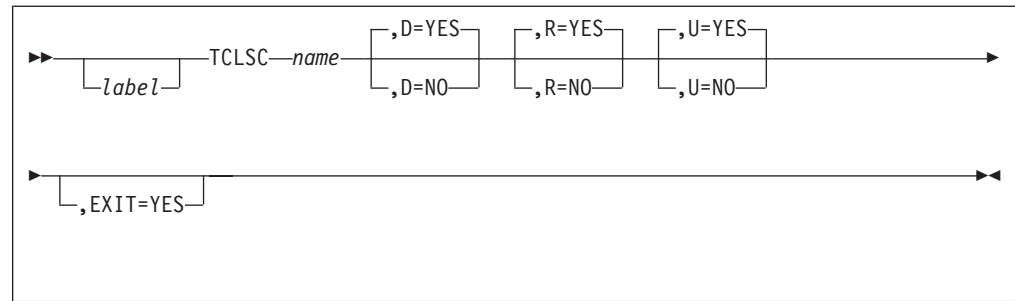
#### **U=YES|NO**

An optional parameter can be coded to indicate whether an unload is to be performed for this tape. If specified, this parameter must be coded as YES or NO. If omitted, a default of YES is assumed.

#### **EXIT=YES**

This parameter is a special interface for EXITC processing. This interface will allow tape close processing to force the tape to be closed and bypass any resulting dumps. This parameter is only valid for the EXITC service routine.

The following positional parameters are still supported:

*name*

A 3-character symbolic general tape name must be specified as the first parameter.

The remaining parameters are described in the preceding list.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must be open when this macro is issued.

## 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 executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
- General tapes that are closed are no longer available to the operational program.
- If the rewind parameter (R) is coded as NO, the tape is closed and positioned before the tape mark which precedes the trailer labels on the current volume of the dataset. If the rewind parameter (R) is coded as YES and the unload parameter (U) is coded as NO, the tape is closed and positioned after the tape mark which follows the header labels on the first volume of the dataset.
- A TCLSC macro or TRSVC macro must be issued for all open tapes prior to issuing an EXITC macro.
- A TCLSC macro performs the equivalent of a TSYNC macro before closing the tape to ensure that all data from a tape buffer is physically written to the tape. (Refer to “TSYNC—Synchronize Tape” on page 490 for a description of synchronization processing.)

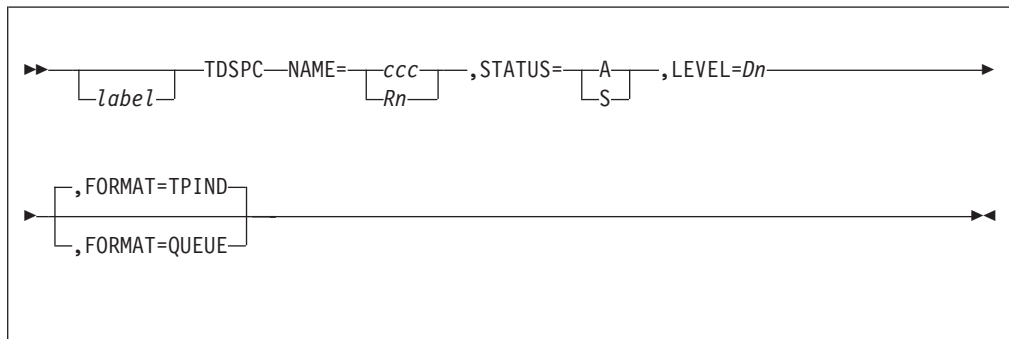
## Examples

None.

## TDSPC–Display Tape Status

The Display Tape Status macro provides the status of the specified tape or the module queue length of a specified active tape.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic general or real-time tape name. It can be:

##### *ccc*

A 3-character string representing a symbolic general or real time tape name. The first two characters must be alphabetic, and the third character must be alphabetic or numeric.

##### (*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.

#### STATUS

Specifies status of the tape. It can be:

**A** For an active tape.

**S** For a standby tape. Do not code **S** with the option **FORMAT=QUEUE**.

#### LEVEL=*Dn*

A symbolic data level (D0–DF) must be specified.

#### FORMAT

Specifies the information returned. It can be:

##### **TPIND**

Returns the status of the tape.

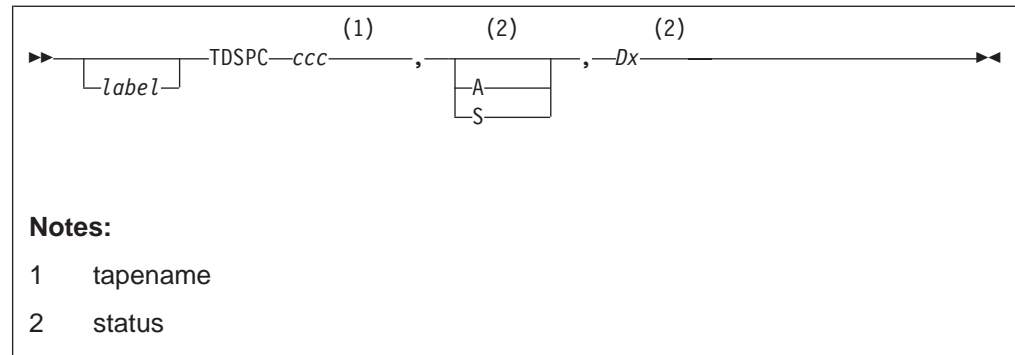
##### **QUEUE**

Returns the module queue length of the tape.

The default is **FORMAT=TPIND**. The **STATUS=S** and **FORMAT=QUEUE** options are not allowed to be specified in the same macro call.

The following macro format is still supported:



*label*

A symbolic name can be assigned to the macro statement.

*tapename*

A 3-character symbolic tape name must be specified as the first parameter. This parameter is required. There is no default.

*status*

Active or standby status must be specified as the second parameter. This parameter must be coded as either A or S. This parameter is required. There is no default.

*data\_level*

A symbolic data level (D0–DF) must be specified as the third parameter. This parameter is required. There is no default.

## 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 R14 are unknown.
- R15 will point to the Tape Status Table entry of the tape name specified.
- The contents of all other registers are preserved across this macro call.
- Certain information from the tape status table is placed in the file address reference word (FARW) on the level specified by this macro.

For **FORMAT=TPIND**, the information is in the following form:

|        |                               |                                                         |
|--------|-------------------------------|---------------------------------------------------------|
| CE1FAx | Byte 0<br>Byte 1<br>Bytes 2–3 | Status indicators<br>Tape chain field<br>Device address |
| CE1FMx | Byte 4                        | Status indicators                                       |
| CE1FCx | Bytes 5–7                     | Tape name                                               |

The bit meanings of the first byte of status indicators are as follows:

| Bit | Meaning                |
|-----|------------------------|
| 0   | Inhibit I/O for device |
| 1   | Inhibit queue restart  |
| 2   | Auxiliary queued       |

## TDSPC

- 3 Standby tape
- 4 Tape is reserved
- 5 Undefined
- 6 Offline
- 7 Closed

The bit meanings of the fifth byte of status indicators are as follows:

| Bit | Meaning                          |
|-----|----------------------------------|
| 0   | Undefined                        |
| 1   | Undefined                        |
| 2   | Tape contains user labels        |
| 3   | Backward tape switch in progress |
| 4   | Awaiting tape ready              |
| 5   | Not first volume                 |
| 6   | Input tape                       |
| 7   | Labels present                   |

For **FORMAT=QUEUE**, the information is in the following form:

|        |                        |                                    |
|--------|------------------------|------------------------------------|
| CE1FAx | Bytes 0–1<br>Bytes 2–3 | Reserved for IBM<br>Device address |
| CE1FMx | Bytes 4–7              | Module queue length                |

## Programming Considerations

1. This macro can be executed on any I-stream.
2. Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
3. If the tape status table does not contain an entry for the specified tape, the FARW is set to zeroes.
4. Any unused bits in the status indicators are set to 0.
5. The 2 bytes of status indicators returned by the **FORMAT=TPIND** form of this macro are not necessarily the primary and secondary status bytes from the tape status table. The definitions of the indicators returned by this macro will remain constant, while the primary and secondary status bytes in the tape status table can be redefined.
6. The **STATUS=S** and **FORMAT=QUEUE** options are not allowed to be specified in the same macro call.

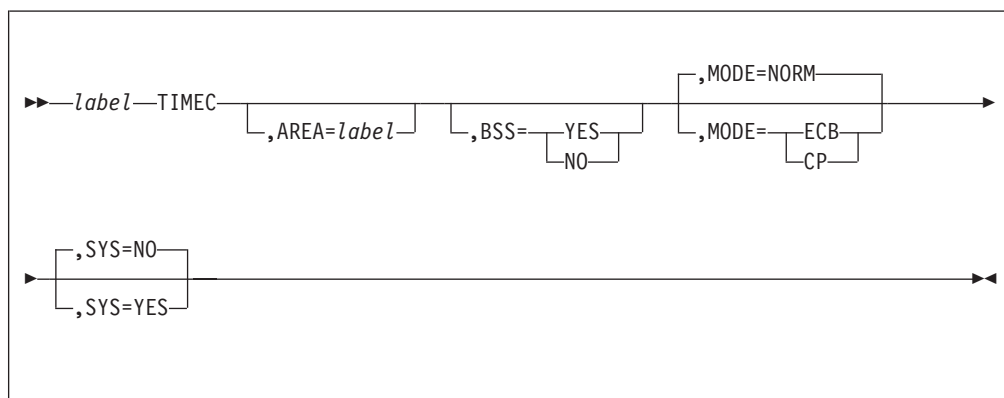
## Examples

None.

## TIMEC—Compute Time Stamp

This macro will return a time stamp in hours-minutes-seconds format for the subsystem local standard time or the system time-of-day clock.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **AREA=*label***

An optional symbolic name of an 8-byte output area where the time stamp is to be placed. If omitted, R5 must contain the address of the output area.

#### **BSS**

This parameter is valid only when the macro is issued by the CP.

##### **NO**

R15 must contain the subsystem index for the SS local time computation.

##### **YES**

R15 will be set to zero and the BSS local time will be computed.

#### **MODE**

Specifies whether a C-Type or E-Type expansion is to be generated. If the MODE parameter is omitted, it defaults to NORM.

##### **NORM**

The TIMEC macro determines the mode from a global variable set by the BEGIN macro.

##### **CP**

A C-Type Expansion is generated.

##### **ECB**

A E-Type Expansion is generated.

#### **SYS**

Specify one of the following:

##### **NO**

The subsystem local time is returned.

##### **YES**

The system local standard time is returned.

If this parameter is omitted, it defaults to NO.

## **TIMEC**

### **Entry Requirements**

- An 8-byte area must be provided to return the computed time stamp. This area may be specified using the AREA parameter or by placing the area address into R5.
- If called by the CP, R15 must contain the subsystem index if the local time is requested (BSS=NO).

### **Return Conditions**

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. R5 contains the address of the user provided 8-byte area. The contents of R7 are unknown if TIMEC was called from the CP. The contents of all other registers are preserved across this macro call.
- The protect key is unchanged.
- The computed time stamp is returned in the user provided 8-byte area. The time is in BCD local standard time in the format:  
hh.mm.ss

Where:

- hh specifies hours in the range 00–23.
- mm specifies minutes in the range 00–59.
- ss specifies seconds in the range 00–59.

### **Programming Considerations**

- This macro can be executed on any I-stream.
- The time value returned by this macro is only updated when the system is above 1052 state.

### **Examples**

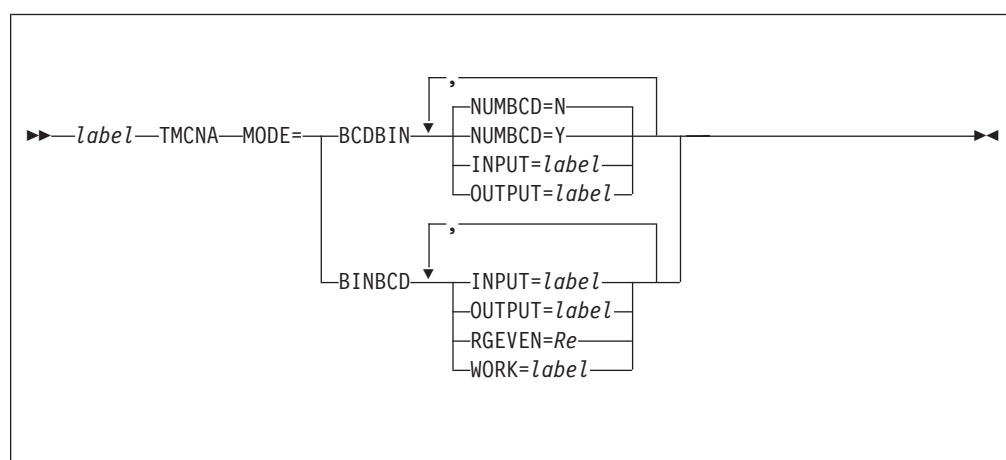
None.

## TMCNA—Time Conversion

This general macro provides three services:

- Conversion of a 2-byte binary number representing minutes since midnight to a 5-byte EBCDIC representation of the time of day based on the 12-hour clock format
- Conversion of a 5-byte EBCDIC representation of the time of day based on the 12-hour clock format to a 2-byte binary number representing minutes since midnight
- Conversion of a 4-byte EBCDIC representation of the time of day based on the 24-hour clock format to a 2-byte binary number representing minutes since midnight.

## Format



*label*

A symbolic name can be assigned to the macro statement.

### MODE=BCDBIN

Indicates that the conversion is from EBCDIC to binary.

### NUMBCD

This parameter specifies whether the EBCDIC input is based on the 12-hour or 24-hour clock.

#### NO

Indicates the 12-hour clock format.

#### YES

Indicates the 24-hour clock format.

NUMBCD is optional and is used only when MODE=BDCBIN is selected.

### INPUT=label

The symbolic address of the field containing the value to be converted.

When NUMBCD=NO, the input parameter addresses a 5-byte field containing the following:

- Byte 0-1 - EBCDIC hour
- Byte 2-3 - EBCDIC minutes
- Byte 4 - an alpha character from the list A, N, P or M representing AM, NOON, PM or Midnight, respectively.

## TMCNA

When NUMBCD=YES, the input parameter addresses a 4-byte field containing the following:

- Byte 0-1 - EBCDIC hour
- Byte 2-3 - EBCDIC minutes

### **OUTPUT=***label*

The symbolic address of a 2-byte field which will contain the number of minutes after midnight upon return from this macro. No alignment is necessary for this field.

### **MODE=BINBCD**

Indicates that the conversion is from binary to EBCDIC.

### **INPUT=***label*

The symbolic address of the field containing the value to be converted. When MODE=BINBCD, this parameter references a 2-byte field containing a binary number less than 1440. No error checking is done on this value.

### **OUTPUT=***label*

The symbolic address of a 5-byte field which will contain the following upon return from this macro.

- Byte 0-1 - EBCDIC hour
- Byte 2-3 - EBCDIC minutes
- Byte 4 An alpha character from the list A, N, P or M representing AM, NOON, PM or Midnight, respectively.

### **RGEVEN=***Re*

The even numbered register of a consecutive even-odd register pair.

### **WORK=***label*

The symbolic address of an 8-byte work area aligned on a doubleword boundary.

## Entry Requirements

None.

## Return Conditions

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

## Programming Considerations

- This macro can be executed on any I-stream.
- The 8-byte field referenced by the WORK parameter must be aligned on a doubleword boundary.
- Both registers of the consecutive even-odd pair referenced by the RGEVEN parameter will be used. The contents of these registers will not be saved.
- Erroneous input data will cause the output from this macro to be invalid. For example, binary input greater than or equal to 1440 causes invalid output.

## Examples

```
ANYNAME TMCNA MODE=BCDBIN,NUMBCD=YES,INPUT=EBW010,OUTPUT=EBW014,
 WORK=EBW024,RGEVEN=R2
```

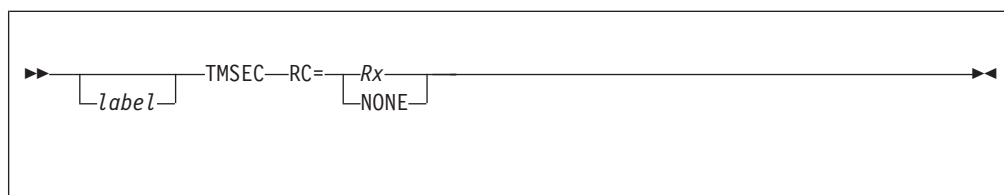
## TMSEC—Epilog for ISO-C Functions Calling TPF Macro Services

This general macro is used with ISO-C support. It is required for writing library functions in assembler language and is the last executable code in the program.

The TMSEC macro completes an interface between non-C language external functions (such as assembly language library routines) and ISO-C programs. The interface begins with a call (in the external function) to the TMSPC macro (normally at the beginning of the program) which preserves the C environment. The TMSEC macro, normally at the end of the program, restores registers after recreating the environment and returns to the calling C language program. In addition, if a stack frame was allocated by TMSPC, it is released by TMSEC.

The TMSPC–TMSEC interface can be used for migrating TARGET(TPF) C library functions to ISO-C.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **RC**

Specifies whether and how a value is returned to the caller. This parameter is required and there is no default.

**Rx** A general purpose register that contains a value to be returned to the calling function. The TMSEC macro passes the return value back to the caller in R15 according to C language protocol.

#### **NONE**

The function does not return a value through a register.

### Entry Requirements

- R9 must contain the address of the current ECB.
- R12 must contain X'2000'. The value was originally set by the call to the TMSPC macro.
- CE3SPTR must contain the address of the current stack frame.
- If a register is specified by the RC parameter, it must contain the return value.

### Return Conditions

Control is returned to the function that originally called the external program containing the TMSPC|TMSEC interface.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro can only be run in the ISO-C environment (where BEGIN is coded with TPFISOC=YES).

## TMSEC

- After TMSEC, the function that called the external function regains control. R13 contains the address of the calling function's stack frame. Registers R2–R14 are restored to the values they contained on entry to the external function. The contents of R0 and R1 cannot be predicted.
- If the RC parameter specifies a register, the value contained in that register is returned to the caller in R15.
- 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.
- There are two ways functions return values depending on the datatype of the value being returned.
  1. If the return value is an integer, a character, or a pointer, the value is returned in the return register.
  2. If the return value is a structure or floating point, a pointer to the value is returned in the first fullword of the C language parameter list.

Functions that return structures (by value) and floating point types do not return their values in registers. Rather, the first fullword of the C language parameter list passed to such a function contains the address of the object into which the function must store the return value.

**Note:** It is the user's responsibility to ensure that the return value (if any) for the external function is returned properly.

## Examples

None.



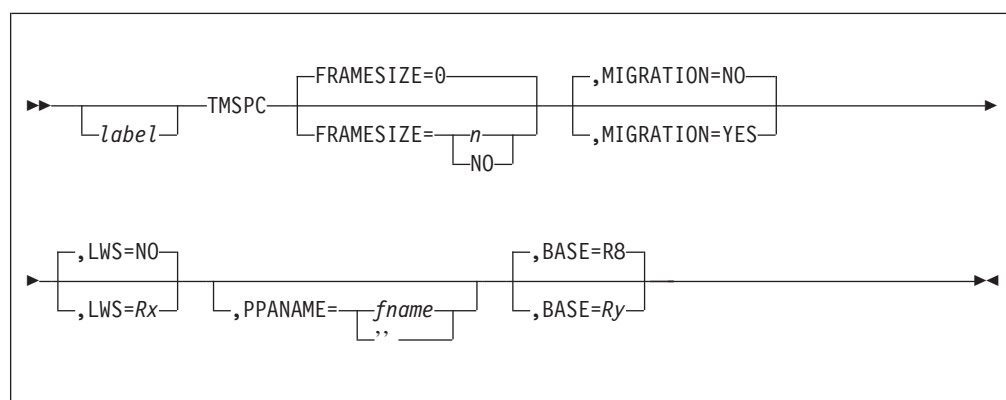
## TMSPC–Prolog for ISO-C Functions Calling TPF Macro Services

This general macro is used with ISO-C support. It is required for writing C library functions in assembler language and must be the first executable code after the BEGIN macro.

The TMSPC macro is the first part of an interface between external assembly language library routines and ISO-C programs. The interface begins with a call (in the external function) to the TMSPC macro (normally at the beginning of the program) preserving the C environment. The TMSEC macro completes the interface. TMSPC generates code that stores all registers in the calling program's ISO-C stack frame. It may allocate a stack frame for the function.

TMSPC can be used to migrate C library functions from TARGET(TPF) to ISO-C.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FRAMESIZE**

Specify one of the following:

- 0** Indicates that the function does not use the C language stack itself but does call another function that requires a stack frame to save and restore registers and the return address. TMSPC allocates the minimum size stack frame. This is the default.
- n** The number of bytes of storage in the stack frame to be reserved for the function's use. This value is added to the size of the minimum stack frame.

#### **NO**

Indicates no stack frame is to be allocated. When FRAMESIZE=NO, TMSPC returns the address of the library work space in Rx.

#### **MIGRATION**

Specify one of the following:

#### **NO**

Uses C language register conventions for the parameter list address (R1) and the current stack frame address (R13). This is the default.

#### **YES**

The parameter list address is loaded into both R1 and R6, and the current

## TMSPC

stack frame address is loaded into both R13 and R7. This reduces the amount of rewriting required to migrate TARGET(TPF) library functions to ISO-C.

### LWS

Specify one of the following:

#### NO

Indicates that no base register is to be loaded with the library workspace (LWS) address. This is the default.

*Rx* Register *Rx* will be loaded with the LWS address and the LWS will be addressable using *Rx*. *Rx* must be a valid base register equate and must not conflict with any other registers set up by the TMSPC macro.

#### PPANAME=*function name*]"

The function name specified is placed in the function name field (PPA1) in TMSPC. If PPANAME is a null string ("), the label for the TMSPC macro is used if there is one. If there is no label, the name of the program calling the TMSPC macro is used. The PPANAME parameter can be useful for debugging or for user exits that process calls to ISO-C functions written in assembler.

### BASE

Specify one of the following:

#### R8

Sets the program base register to R8. This is the default.

*Ry* Sets the user-specified program base register.

## Entry Requirements

- Type 1 register conventions are assumed.
- Register R1 points to the C language parameter list if parameters are passed or if the return value has a datatype structure or floating point. Otherwise, the contents of R1 are unknown.
- R12 must point to the C environment's task communications area (TCA).
- R13 must point to the caller's data save area (DSA).
- R14 must contain the return address.
- R15 must contain the function base address.
- The contents of all other registers are unknown.

## Return Conditions

- If the MIGRATION parameter is coded as (or defaults to) NO, R1 points to the C language parameter list, if there is one. If MIGRATION=YES is coded, R6 points to the parameter list.
- If FRAMESIZE=NO and
  - MIGRATION=NO, R13 and CE3SPTR point to the caller's stack frame.
  - MIGRATION=YES, R7 also points to the caller's stack frame.
- If FRAMESIZE is a number and MIGRATION=NO, R13 and CE3SPTR point to the newly created stack frame. The address of the LWS is not loaded. If MIGRATION=YES is coded, R7 also points to the newly created stack frame.
- If the LWS parameter is coded, the register specified points to the LWS.
- The register specified by the BASE parameter is used as the base register for the function.
- R8 points to the program header of the function.

- R9 contains the ECB address.
- R11 points to LOCORE page 1 (X'1000').
- R12 points to LOCORE page 2 (X'2000').
- If FRAMESIZE is coded with a number, tags are provided in the new stack frame for CSTKLBAS and CSTKMIN. The base register for CSTKLBAS is R7 if MIGRATION=YES or R13 if MIGRATION=NO. (CSTKMIN is an absolute symbol.)
- Register R15 is dropped as a base register.

## Programming Considerations

- This macro can be run on any I-stream.
- This macro can be issued more than once in a particular module, but each instance of the macro must be followed by at least one TMSEC macro before another TMSPC macro can be issued. This grouping of TMSPC and TMSEC macros allows multiple functions to be coded within a single function module. See "TMSEC—Epilog for ISO-C Functions Calling TPF Macro Services" on page 399 for more information about the TMSEC macro.
- The caller must provide a DSA pointed to by R13. If a requested stack frame does not fit in the current stack allocation, the ISO-C Stack Overflow Routine in the control program is activated. If the Stack Overflow Routine is unable to acquire a stack block, the appropriate dump is issued and the ECB exits.

## Examples

- The following example creates a prolog that saves all registers, creates a new stack frame with 66 bytes of storage for the function, and sets up the registers to allow calls to TPF macro services:

```
TMSPC FRAMESIZE=66
```

- The following example creates a prolog that allocates a minimal stack frame and sets up R6 and R7 according to TARGET(TPF) register conventions:

```
TMSPC FRAMESIZE=0,MIGRATION=YES
```

- The following example creates a prolog that does not allocate a new stack frame and loads the address of the library work space in R2.

```
TMSPC FRAMESIZE=NO,LWS=R2
```

- The following example shows a TARGET(TPF) C library function migrated ISO-C using TMSPC and TMSEC.

```

* TARGET(TPF) version of C library function. *

 BEGIN NAME=CFUN (point a)
 ICPLOG HIGHREG=R2,FRAMESIZE=16 (point b)
***** Initial processing *****
 ST R8,CSTKLBAS Save my R8 on the stack.
*
 L R8,CEISVP Load library caller's R8.
***** Call TFP macro *****
 L R8,CSTKLBAS Restore my R8.
***** Final processing *****
 L R6,returncode
 ICELOG (point c)
 FINIS
 END

* TPF ISO-C version of C library function. *

 BEGIN NAME=CFUN,TPFISOC=YES (point a)
 TMSPC FRAMESIZE=16,MIGRATION=YES (point b)
```

## TMSPC

```

***** Initial processing *****
 ST R8,CSTKLBAS Save my R8 on the stack.
*
 L R8,CE1SVP Load library caller's R8.
***** Call TFP macro *****
 L R8,CSTKLBAS Restore my R8.
*
***** Final processing *****
 L R6,returncode
 TMSEC RC=R6 (point c)
 FINIS
 END

```

- At point a, the ISO-C version adds the TPFISOC=YES parameters to the BEGIN macro, indicating that the function will be link-edited into an ISO-C load module and will be called from the ISO-C environment.
  - At point b, the ISO-C version changes ICPLOG to TMSPC, deletes the HIGHREG parameter (TMSPC saves all of the caller's registers), and adds the MIGRATION=YES parameter to specify TARGET(TPF) register conventions for the PLIST address (R6) and the current stack frame (R7).
  - At point c, the ISO-C version changes ICELOG to TMSEC and adds the RC=R6 parameter.
- The following example is an ISO-C library program that calls a TPF macro to put a time stamp in the user expansion area of the stack frame. The ISO-C program returns a pointer to the time stamp as its value.

```

 PRINT NOGEN
 BEGIN NAME=C001,VERSION=41,TPFISOC=YES
*
 TMSPC FRAMESIZE=NO,LWS=R6 Pointer to LWS in R6. No frame.
 L R5,ILWSUEXP Address of User Expansion Area
 ST R8,ILWSLBAS Save the library base
 L R8,CE1SVP Get the Application base
 TIMEC , Get the timestamp into UEXP
 L R8,ILWSLBAS Restore library base
 TMSEC RC=R5 Return to C application
*
 LTORG
 FINIS
 END

```

- The following is an example of two ISO-C library functions that reside in a single library program.

```

 BEGIN NAME=CKEYRC,IBM=YES,VERSION=40,TPFISOC=YES

* Entry point for 'keyrc()' function. This is the normal KEYRC and *
* sets the protection key to the key of the ECB, which is a key of 1.*

 TMSPC FRAMESIZE=30,PPANAME=keyrc
 L R8,CE1SVP Get the Application base
 KEYRC
 TMSEC RC=NONE
 LTORG

* Entry point for 'keyrc_okey()' function. This is the entry point *
* for the special KEYRC OKEY=YES and sets the protection key to the *
* one saved by the previous CINFC W call. *

@@OKEYRC TMSPC FRAMESIZE=30,PPANAME=keyrc_okey
 L R8,CE1SVP Get the Application base

```

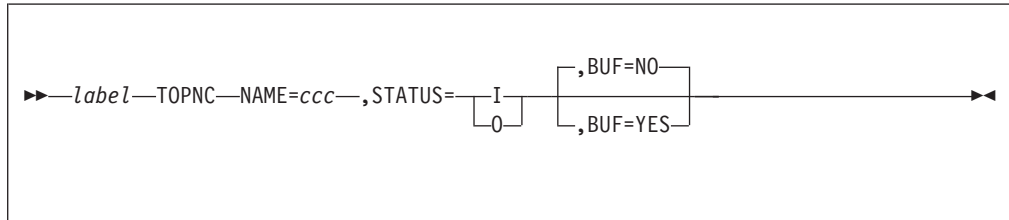
```
KEYRC OKEY=YES
TMSEC RC=NONE
LTORG
```

```
FINIS
END
```

## TOPNC—Open a General Tape

This general macro makes the specified general tape available to the operational program.

### 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 2 characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first two characters cannot be RT.

**(R*n*)**

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.

#### STATUS=**I|O**

The input or output status must be specified. This parameter must be coded as either I or O.

#### BUF

An optional keyword parameter can be specified indicating the output mode to be used when writing to buffered devices.

##### YES

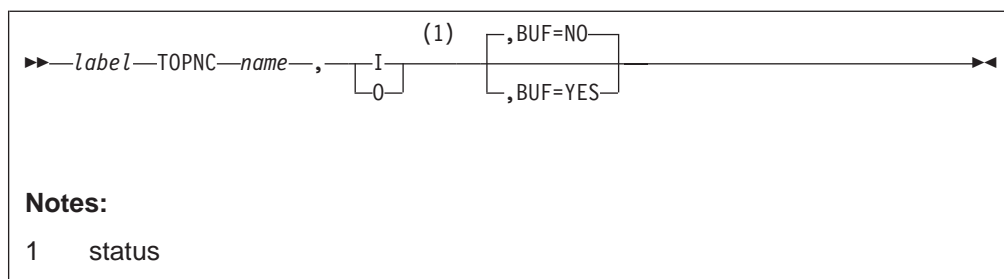
Buffered mode is to be used.

##### NO

Tape Write Immediate (TWI) mode is to be used.

If omitted, a default of BUF=NO is assumed. This parameter has meaning only when the tape is mounted on a buffered device. It is ignored when the tape is mounted on a nonbuffered device.

The following macro format is still supported:

**label**

A symbolic name can be assigned to the macro statement.

**name**

A 3-character symbolic general tape name must be specified as the first parameter.

**status**

The input or output status must be specified as the second parameter. This parameter must be coded as either I or O.

**BUF**

An optional keyword parameter can be specified indicating the output mode to be used when writing to buffered devices.

**YES**

Buffered mode is to be used.

**NO**

Tape Write Immediate (TWI) mode is to be used.

If omitted, a default of BUF=NO is assumed. This parameter has meaning only when the tape is mounted on a buffered device. It is ignored when the tape is mounted on a nonbuffered device.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must not be open when this macro is issued.

## 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.
- Control is returned to the operational program when the specified tape is ready for use.
- The positioning of the specified tape is unchanged. (Refer to “TCLSC—Close a General Tape” on page 390 for more information on tape positioning after a tape is closed.)

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**

## TOPNC

- A TCLSC macro or TRSVC macro must be issued for all open tapes prior to issuing an EXITC macro.
- If the tape is mounted on a nonbuffered device, the buffered mode keyword parameter (BUF) is ignored.
- If the tape is opened as an output tape and is mounted in blocked mode, the buffered mode keyword parameter (BUF) causes the appropriate indicator to be set in the tape status table entry. However, this indicator will be ignored for subsequent writes to the tape, and all write operations will be performed in buffered mode.
- If this macro is issued prior to the end of tape restart, the ECB is exited and a system error issued.

## Examples

None.



## TOURC—Write a Real-Time Tape Record and Release Core Block

This general macro will write a record contained in a storage block to a real-time tape. It will also return the storage block to the appropriate pool and make it unavailable to the operational program.

### Format

```
►►—label—TOURC—NAME=ccc—,LEVEL=Dx—◄◄
```

#### *label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic real-time tape name. It can be:

#### *ccc*

A 3-character string representing a symbolic real-time tape name. The first 2 characters must be alphabetic, and the third character must be alphabetic or numeric.

#### (*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.

The following macro format is still supported:

```
►►—label—TOURC—name—,—Dx—◄◄
```

#### *label*

A symbolic name can be assigned to the macro statement.

#### *name*

A 3-character symbolic real-time tape name must be specified as the first parameter.

*Dx* 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.
- A storage block must be held by the ECB on the data level specified by this macro.

### Return Conditions

- Control is returned to the next sequential instruction.

## TOURC

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- The storage block containing the data record is no longer available to the operational program.
- The file address reference word on the data level specified by this macro is unchanged.
- The core block reference word on the data level specified by this macro is updated to indicate that the storage block is no longer held by the ECB.

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
- The control program checks to determine if the ECB is holding a storage block on the data level specified by this macro. If no block is held, control is transferred to the system error routine.
- The status of the Write operation can never be determined by the operational program.
- The operational program can use the request level specified by this macro immediately upon return from the control program.
- The contents of the entire storage block are written to tape.
- The record written to tape will have 16 bytes of appended data containing the subsystem name, subsystem user name, and value of the TOD clock.

**Note:** The value of the TOD clock will be the time at which the macro was issued.

- If the tape is mounted on a buffered device, the operation is performed in buffered mode.
- If this macro is issued prior to the end of tape restart, the ECB is exited and a system error issued.

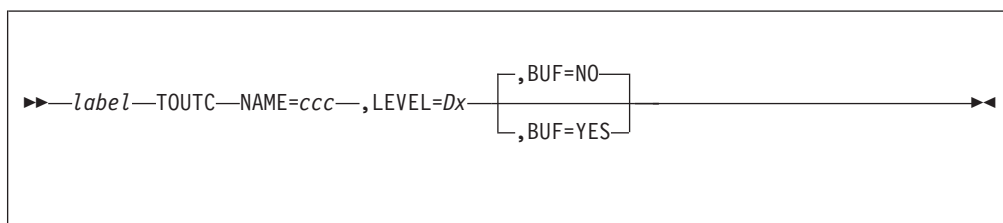
## Examples

None.

## TOUTC—Write a Real-Time Tape Record

This general macro will write a record contained in a storage block to a real-time tape.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **NAME**

Specifies the symbolic real-time tape name. It can be:

#### *ccc*

A 3-character string representing a symbolic real-time tape name. The first 2 characters must be alphabetic and the third character must be alphabetic or numeric.

#### **(R*n*)**

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=D*x***

A symbolic data level (D0–DF) must be specified.

#### **BUF**

An optional keyword parameter can be specified indicating the output mode to be used when writing to buffered devices.

#### **YES**

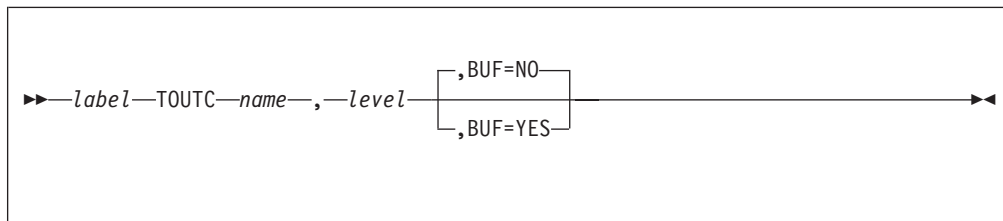
Buffered mode is to be used.

#### **NO**

Tape Write Immediate (TWI) mode is to be used.

If omitted, a default of BUF=NO is assumed. This parameter has meaning only when the tape is mounted on a buffered device. It is ignored when the tape is mounted on a nonbuffered device.

The following macro format is still supported:



#### *label*

A symbolic name can be assigned to the macro statement.

## TOUTC

### *name*

A 3-character symbolic real-time tape name must be specified as the first parameter.

### *level*

A symbolic data level (D0–DF) must be specified as the second parameter.

### **BUF**

An optional keyword parameter can be specified indicating the output mode to be used when writing to buffered devices.

#### **YES**

Buffered mode is to be used.

#### **NO**

Tape Write Immediate (TWI) mode is to be used.

If omitted, a default of BUF=NO is assumed. This parameter has meaning only when the tape is mounted on a buffered device. It is ignored when the tape is mounted on a nonbuffered device.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The first 4 bytes of the file address reference word (FARW) of the data level specified by this macro must contain the address of the data. The record to be written must be above X'1000'. The last 2 bytes of the FARW must contain the byte count of the record. This byte count must be in the range 1–32 752.

## 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 status of the Write operation is unknown.
- The FARW on the data level specified by this macro is unchanged.

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
- Tape data transfer depends on various conditions:
  - For a blocked tape, a WAITC 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 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.
- The record written to tape will have 16 bytes of appended data containing the subsystem name, subsystem user name, and value of the time-of-day clock.

**Note:** The value of the time-of-day clock will be the time at which the macro was issued.

- The storage area containing the data that is being written must remain unchanged until the operation is complete.
- This macro should not be used to write Heap storage areas to a blocked tape.
- If this macro is issued prior to the end of tape restart, the ECB is exited and a system error issued.

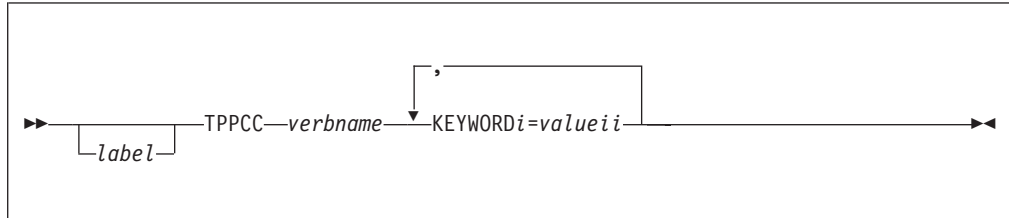
## **Examples**

None.

## TPPCC–TPF/APPC Conversation Verb Macro

Use this macro to provide the interface for all conversation verbs defined in the LU 6.2 architecture that TPF supports. This section contains a description of the TPPCC macro in its general form, followed by a separate section for each of the valid verbs.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### *verbname*

Specifies the name of the TPF/APPC verb to be executed. The valid *verbnames* are shown in Table 10 on page 415.

#### **KEYWORDi=valuei**

Specifies the first valid keyword parameter for the *verbname* specified.

Each macro call begins with TPPCC, followed by the positional parameter *verbname*, which specifies the verb to be executed. This is followed by one or more keyword parameters and values for those parameters. The values specify either a keyword option or a main storage location of a field.

The main storage location of a field can be specified as either the symbolic name of the field or as a register that points to the field. If you specify a register, the register name must be enclosed in parentheses and must be in the range R1–R7.

Table 10 on page 415 is a summary of the supported verb names and their associated parameters. The table also contains:

- An indication of whether the parameter is passed to or returned from the TPF/APPC processing components
- A brief description of the parameter
- The equivalent parameter defined in the LU 6.2 architecture.

More detailed information for each parameter is provided in the individual verb descriptions later in this chapter.

Table 10. TPF/APPC Conversation Verbs and Valid Keywords

| Verb Name                                                                                                                                                   | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                                 | Architecture Equivalent |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| ACTIVATE_ON_CONFIRMATION                                                                                                                                    | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                             | RESOURCE                |
|                                                                                                                                                             | VERB              | Passed             | This specifies the architecture's verb function to be performed in conjunction with this verb. Valid choices are: <ul style="list-style-type: none"> <li>• CONFIRM</li> <li>• DEALLOCATE</li> <li>• PREPARE_TO_RECEIVE</li> </ul>           | None                    |
|                                                                                                                                                             | PARM              | Passed             | This specifies the data level (DL) indication or the TOKEN to be passed to the activated program.                                                                                                                                           | None                    |
|                                                                                                                                                             | PGM               | Passed             | This specifies the TPF E-type program to be activated when there is data or other information available to satisfy the CONFIRM request. The program specified must be defined in the TPF transaction program name table (TPNT) as TYPE=AOR. | None                    |
|                                                                                                                                                             | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                     | RETURN_CODE             |
| <b>Note:</b> The ACTIVATE_ON_CONFIRMATION verb is a TPF-only extension to the LU 6.2 architecture. There is no equivalent verb defined in the architecture. |                   |                    |                                                                                                                                                                                                                                             |                         |
| ACTIVATE_ON_RECEIPT                                                                                                                                         | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                             | RESOURCE                |
|                                                                                                                                                             | PARM              | Passed             | This specifies the data level (DL) indication or the TOKEN to be passed to the activated program.                                                                                                                                           | None                    |
|                                                                                                                                                             | PGM               | Passed             | This specifies the TPF E-type program to be activated when there is data or other information available to satisfy the RECEIVE request. The program specified must be defined in the TPF TPNT as TYPE=AOR.                                  | None                    |
|                                                                                                                                                             | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                     | RETURN_CODE             |
| <b>Note:</b> The ACTIVATE_ON_RECEIPT verb is a TPF-only extension to the LU 6.2 architecture. There is no equivalent verb defined in the architecture.      |                   |                    |                                                                                                                                                                                                                                             |                         |
| ALLOCATE                                                                                                                                                    | LUNAME            | Passed             | This specifies the fully qualified name of the remote LU.                                                                                                                                                                                   | LU_NAME                 |
|                                                                                                                                                             | TPN               | Passed             | This specifies the name of the remote transaction program used on this conversation.                                                                                                                                                        | TPN                     |

Table 10. TPF/APPC Conversation Verbs and Valid Keywords (continued)

| Verb Name | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                                                                                                                                                                                | Architecture Equivalent |
|-----------|-------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
|           | RCONTROL          | Passed             | This specifies the condition on which control is returned to the issuing program. The WSA option specifies to return control when a session is allocated for use by the conversation. The IMM option specifies to return control immediately if no contention-winner session is currently active and available. TPF does not support the other options defined by the LU 6.2 architecture. | RETURN_CONTROL          |
|           | MODE              | Passed             | This specifies the name used to designate the properties of the session to be allocated.                                                                                                                                                                                                                                                                                                   | MODE_NAME               |
|           | TYPE              | Passed             | This specifies the type of conversation. The BASIC option provides support for the BASIC_CONVERSATION option defined by the LU 6.2 architecture. The MAPPED option provides support for the MAPPED_CONVERSATION option defined by the LU 6.2 architecture.                                                                                                                                 | TYPE                    |
|           | PIP               | Passed             | This specifies whether or not program initialization parameters are supported. NO is the only supported option. TPF does not support the YES option defined by the LU 6.2 architecture.                                                                                                                                                                                                    | PIP                     |
|           | SYNC              | Passed             | This specifies the synchronization level allowed on the allocated conversation. Only the NONE and CONFIRM options are supported. TPF does not support the SYNCPT option defined by the LU 6.2 architecture.                                                                                                                                                                                | SYNC_LEVEL              |
|           | SECURITY          | Passed             | This specifies the security information used to verify the identity of the end users of the conversation. The option NO provides support for the LU 6.2 architecture's NONE option. This indicates that access security information is omitted on this conversation. TPF does not support the architecture's SAME and PGM options.                                                         | SECURITY                |
|           | RESID             | Returned           | This specifies where the resource ID is to be returned. The resource ID uniquely identifies this conversation from all others and is used when issuing other TPF/APPC verbs for this conversation. The resource ID is 4 bytes long.                                                                                                                                                        | RESOURCE                |
|           | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                    | RETURN_CODE             |
| CONFIRM   | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                            | RESOURCE                |
|           | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                    | RETURN_CODE             |



Table 10. TPF/APPC Conversation Verbs and Valid Keywords (continued)

| Verb Name      | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                  | Architecture Equivalent     |
|----------------|-------------------|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
|                | RTSRCVD           | Returned           | This specifies where the REQUEST_TO_SEND indication is to be returned.                                                                                                                                                       | REQUEST_TO_SEND_RECEIVED    |
| CONFIRMED      | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                              | RESOURCE                    |
|                | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                      | RETURN_CODE                 |
| DEALLOCATE     | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                              | RESOURCE                    |
|                | TYPE              | Passed             | This specifies the type of deallocation to be done.                                                                                                                                                                          | TYPE                        |
|                | LOGDATA           | Passed             | This specifies whether or not error information is to be logged. NO is the only option supported. TPF does not support the YES option defined by the LU 6.2 architecture.                                                    | LOG_DATA                    |
|                | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                      | RETURN_CODE                 |
| FLUSH          | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                              | RESOURCE                    |
|                | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                      | RETURN_CODE                 |
| GET_ATTRIBUTES | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                              | RESOURCE                    |
|                | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                      | RETURN_CODE                 |
|                | OWNAME            | Returned           | This specifies where the local LU name is returned.                                                                                                                                                                          | OWN_FULLY_QUALIFIED_LU_NAME |
|                | PLUNAME           | Returned           | This specifies where the name of the partner LU is returned. The partner LU name and the fully qualified partner LU name are identical.                                                                                      | PARTNER_LU_NAME             |
|                | MODE              | Returned           | This specifies where the mode name used for this conversation is returned.                                                                                                                                                   | MODE_NAME                   |
|                | SYNC              | Returned           | This specifies where the synchronization level used for this conversation is returned. Only the synchronization levels NONE and CONFIRM supported. TPF does not support the SYNCPT level defined by the LU 6.2 architecture. | SYNC_LEVEL                  |
| GET_TYPE       | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                              | RESOURCE                    |
|                | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                      | RETURN_CODE                 |

Table 10. TPF/APPC Conversation Verbs and Valid Keywords (continued)

| Verb Name          | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Architecture Equivalent |
|--------------------|-------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
|                    | TYPE              | Returned           | This specifies where the conversation type is returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | TYPE                    |
| POST_ON_RECEIPT    | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | RESOURCE                |
|                    | FILL              | Passed             | This specifies when posting should occur based on the logical record format. Only the LL option is supported, which specifies that posting occurs when a complete or truncated logical record is received, or when a part of a logical record is received that is at least equal in length to that specified on the LENGTH parameter. TPF does not support the BUFFER option defined by the LU 6.2 architecture.                                                                                                                                                                           | FILL                    |
|                    | LENGTH            | Passed             | This specifies the minimum amount of data that the LU must receive before the conversation can be posted. This parameter is used along with the FILL parameter to determine when to post the conversation.                                                                                                                                                                                                                                                                                                                                                                                 | LENGTH                  |
|                    | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | RETURN_CODE             |
| PREPARE_TO_RECEIVE | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | RESOURCE                |
|                    | TYPE              | Passed             | This specifies the type of PREPARE_TO_RECEIVE to be performed. The FLUSH, CONFIRM and SYNC options provide support for the LU 6.2 architecture's FLUSH, CONFIRM, and SYNC_LEVEL options, respectively. If FLUSH or CONFIRM is specified, the appropriate verb function is performed. If SYNC is specified, the synchronization level specified on the ALLOCATE is used to determine which verb function is to be performed. If the synchronization level was NONE, the FLUSH verb function is performed. If the synchronization level was CONFIRM, the CONFIRM verb function is performed. | TYPE                    |
|                    | LOCKS             | Passed             | This specifies when control is to be returned, when the CONFIRM verb is implied. Only the SHORT option is supported, which causes control to be returned upon the receipt of a positive reply to CONFIRM. TPF does not support the LONG option defined by the LU 6.2 architecture.                                                                                                                                                                                                                                                                                                         | LOCKS                   |
|                    | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | RETURN_CODE             |
| RECEIVE            | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | RESOURCE                |

Table 10. TPF/APPC Conversation Verbs and Valid Keywords (continued)

| Verb Name       | Keyword Parameter | Passed or Returned  | Description                                                                                                                                                                                                                                                                                                                                                                             | Architecture Equivalent  |
|-----------------|-------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
|                 | FILL              | Passed              | This specifies when the receive should be satisfied based on the logical record format. Only the LL option is supported, which specifies the program is to receive one complete or truncated logical record, or a portion of a logical record that is equal to the length specified by the LENGTH parameter. TPF does not support the BUFFER option defined by the LU 6.2 architecture. | FILL                     |
|                 | WAIT              | Passed              | This specifies that the RECEIVE_AND_WAIT verb function should be performed. TPF does not support RECEIVE_IMMEDIATE verb defined by the LU 6.2 architecture.                                                                                                                                                                                                                             | RECEIVE_AND_WAIT verb    |
|                 | LENGTH            | Passed and Returned | This specifies the maximum length of data that the TP can receive. When control is returned to the TP, this variable contains the actual amount of data the program received up to the maximum. If the program receives information other than data, this variable remains unchanged.                                                                                                   | LENGTH                   |
|                 | DATA              | Passed              | This specifies the storage address in which the program is to receive the data.                                                                                                                                                                                                                                                                                                         | DATA                     |
|                 | RCODE             | Returned            | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                 | RETURN_CODE              |
|                 | RTSRCVD           | Returned            | This specifies where the REQUEST_TO_SEND indication is to be returned.                                                                                                                                                                                                                                                                                                                  | REQUEST_TO_SEND_RECEIVED |
|                 | WHATRCV           | Returned            | This specifies where the WHAT_RECEIVED indication is returned. The WHAT_RECEIVED indication may indicate data, confirmation, or conversation status.                                                                                                                                                                                                                                    | WHAT_RECEIVED            |
| REQUEST_TO_SEND | RESID             | Passed              | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                         | RESOURCE                 |
|                 | RCODE             | Returned            | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                 | RETURN_CODE              |
| SEND_DATA       | RESID             | Passed              | This specifies the resource ID returned by the ALLOCATE verb, or the resource ID assigned by an incoming ATTACH.                                                                                                                                                                                                                                                                        | RESOURCE                 |
|                 | DATA              | Passed              | This specifies the address of the data to be sent.                                                                                                                                                                                                                                                                                                                                      | DATA                     |
|                 | LENGTH            | Passed              | This specifies the length of the data to be sent. This data length is in no way related to the length of a logical record. It is used only to determine the length of the data located at the address specified on the DATA parameter.                                                                                                                                                  | LENGTH                   |
|                 | RCODE             | Returned            | This specifies where the return code is to be returned.                                                                                                                                                                                                                                                                                                                                 | RETURN_CODE              |

Table 10. TPF/APPC Conversation Verbs and Valid Keywords (continued)

| Verb Name                                                                                                                                                                                                                                                                  | Keyword Parameter | Passed or Returned | Description                                                                                                                                                                                                                | Architecture Equivalent  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
|                                                                                                                                                                                                                                                                            | RTSRCVD           | Returned           | This specifies where the REQUEST_TO_SEND indication is to be returned.                                                                                                                                                     | REQUEST_TO_SEND_RECEIVED |
| SEND_ERROR                                                                                                                                                                                                                                                                 | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                            | RESOURCE                 |
|                                                                                                                                                                                                                                                                            | TYPE              | Passed             | This specifies the type of error that has been detected. The PROG option specifies that the transaction program detected an error. The SVC option specifies that a TPF service program detected an error.                  | TYPE                     |
|                                                                                                                                                                                                                                                                            | LOGDATA           | Passed             | This specifies whether or not error information is to be logged. Only the NO option is supported. TPF does not support the YES option defined by the LU 6.2 architecture.                                                  | LOG_DATA                 |
|                                                                                                                                                                                                                                                                            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                    | RETURN_CODE              |
|                                                                                                                                                                                                                                                                            | RTSRCVD           | Returned           | This specifies where the REQUEST_TO_SEND indication is to be returned.                                                                                                                                                     | REQUEST_TO_SEND_RECEIVED |
| TEST                                                                                                                                                                                                                                                                       | RESID             | Passed             | This specifies the resource ID returned by the ALLOCATE verb or the resource ID assigned by an incoming ATTACH.                                                                                                            | RESOURCE                 |
|                                                                                                                                                                                                                                                                            | TEST              | Passed             | This specifies which condition is to be tested. The POSTED option specifies to test whether the conversation has been posted. The RTSRCVD option specifies to test whether a REQUEST_TO_SEND indication has been received. | TEST                     |
|                                                                                                                                                                                                                                                                            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                    | RETURN_CODE              |
| WAIT                                                                                                                                                                                                                                                                       | RESIDL            | Passed             | This specifies where a list of resource IDs are located.                                                                                                                                                                   | RESOURCE_LIST            |
|                                                                                                                                                                                                                                                                            | RCODE             | Returned           | This specifies where the return code is to be returned.                                                                                                                                                                    | RETURN_CODE              |
|                                                                                                                                                                                                                                                                            | RESPSTD           | Returned           | This specifies where the resource ID of the conversation posted is returned.                                                                                                                                               | RESOURCE_POSTED          |
| <b>Note:</b><br><b>Passed</b> Indicates that the value or option is passed from the transaction program to the verb processing component.<br><b>Returned</b> Indicates that the value or option is returned to the transaction program from the verb processing component. |                   |                    |                                                                                                                                                                                                                            |                          |

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The conversation must be in a valid state for the particular *verbname* specified. The valid states for each verb are listed in the “Entry Requirements” section for that verb. See the *TPF ACF/SNA Data Communications Reference* for more information about the finite state machine and valid states.
- Other entry requirements are dependent on the *verbname* specified. See the *verbname* section for specific information.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R14 and R15 are unknown. The contents of R0–R7 are preserved across this macro call.
- The TPF/APPC support always returns to the calling program with a protection key of 1.
- Other return conditions are dependent on the *verbname* specified. See the *verbname* section for specifics.
- The TPPCC macro returns a 6-byte return code. The first 2 bytes are the primary return code, and the next 4 bytes are the secondary return code.

The primary return code contains the value zero for normal return. In general, when the primary return code is zero, the secondary return code has no meaning. The following verbs are exceptions to this; for these verbs the secondary return code does contain meaningful information on a normal return:

ACTIVATE\_ON\_CONFIRMATION  
 ACTIVATE\_ON\_RECEIPT  
 TEST  
 WAIT

The primary return code contains a nonzero value for an error return. In general, the secondary return code provides more detailed error information. Again, there are exceptions to this; some of the primary return codes do not have a secondary code associated with it.

Table 11 contains a list of all the primary return codes and their meanings, and Table 12 on page 422 contains a list of all the secondary return codes and their meanings. Not all return codes are possible for every verb. The individual TPF/APPC verb descriptions document the valid return codes for that TPF/APPC verb. The symbolic names of the return code values are defined in macro TPPCE.

Table 11. TPPCC Primary Return Codes

| Symbolic Name          | Hex Value | Meaning                                                             |
|------------------------|-----------|---------------------------------------------------------------------|
| LU62RC_OK              | 0000      | The function completed successfully.                                |
| LU62RC_PARAMETER_CHECK | 0001      | An invalid parameter was detected. Check the secondary return code. |
| LU62RC_STATE_CHECK     | 0002      | An invalid state was detected. Check the secondary return code.     |
| LU62RC_ALLOC_ERROR     | 0003      | An allocation error occurred. Check the secondary return code.      |
| LU62RC_DLLOC_ABEND_PGM | 0006      | An error occurred, and the conversation was deallocated.            |

## TPPCC

Table 11. TPPCC Primary Return Codes (continued)

| Symbolic Name            | Hex Value | Meaning                                                                                                    |
|--------------------------|-----------|------------------------------------------------------------------------------------------------------------|
| LU62RC_DLLOC_ABEND_SVC   | 0007      | An error occurred, and the conversation was deallocated.                                                   |
| LU62RC_DLLOC_ABEND_TMR   | 0008      | An error occurred, and the conversation was deallocated.                                                   |
| LU62RC_DLLOC_NORMAL      | 0009      | The conversation was deallocated.                                                                          |
| LU62RC_POSTING_NOTACTIV  | 000B      | Posting is not active for the specified conversation.                                                      |
| LU62RC_PGMERR_NOTRUNC    | 000C      | A program error occurred. Data was not truncated.                                                          |
| LU62RC_PGMERR_TRUNC      | 000D      | A program error occurred. Data was truncated.                                                              |
| LU62RC_PGMERR_PURGING    | 000E      | A program error occurred. Data was purged.                                                                 |
| LU62RC_CONVFAIL_RETRY    | 000F      | The conversation failed due to a session outage.                                                           |
| LU62RC_CONVFAIL_NORETRY  | 0010      | The conversation failed due to a protocol error.                                                           |
| LU62RC_SVCERR_NOTRUNC    | 0011      | A service program error occurred. Data was not truncated.                                                  |
| LU62RC_SVCERR_TRUNC      | 0012      | A service program error occurred. Data was truncated.                                                      |
| LU62RC_SVCERR_PURGING    | 0013      | A service program error occurred. Data was purged.                                                         |
| LU62RC_LLRCV_UNSUCESSFUL | 0014      | The condition checked by the TEST verb was not set. See the TEST verb description for further information. |
| LU62RC_ALLOC_UNSUCESSFUL | 0015      | The ALLOCATE verb with RCONTROL=IMM was not able to allocate a session to this conversation immediately.   |
| LU62RC_TPF_ABEND         | FFFF      | This indicates a system abend or time-out.                                                                 |

Table 12. TPPCC Secondary Return Codes

| Symbolic Name           | Hex Value | Meaning                                                                                                                                            |
|-------------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| LU62RC_POSTED_DATA      | 00000000  | The conversation was posted. The information available is data.                                                                                    |
| LU62RC_POSTED_NOTDATA   | 00000001  | The conversation was posted. The information available is <b>not</b> data.                                                                         |
| LU62RC_PK_BAD_TCBID     | 00000001  | A parameter check was detected. The transaction control block (TCB) ID is invalid.                                                                 |
| LU62RC_PK_BAD_CONVID    | 00000002  | A parameter check was detected. The resource ID is invalid.                                                                                        |
| LU62RC_PK_EMPTY_RESIDL  | 00000003  | A parameter check was detected. The resource list specified on the WAIT verb is empty.                                                             |
| LU62RC_ALLOCERR_NORETRY | 00000004  | An allocation error occurred. Do not retry the request.                                                                                            |
| LU62RC_ALLOCERR_RETRY   | 00000005  | An allocation error occurred. Retry the request.                                                                                                   |
| LU62RC_INVALID_LENGTH   | 00000006  | A parameter check was detected. The LENGTH parameter coded on the POST_ON_RECEIPT, RECEIVE, or SEND_DATA verb exceeds the maximum value of 32,767. |

Table 12. TPPCC Secondary Return Codes (continued)

| Symbolic Name           | Hex Value | Meaning                                                                                                                                                                                                                                         |
|-------------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LU62RC_AOC_INCOMPLETE   | 00000009  | The ACTIVATE_ON_CONFIRMATION verb was accepted. The program specified (PGM) is activated when the CONFIRM function is complete.                                                                                                                 |
| LU62RC_AOR_INCOMPLETE   | 00000009  | The ACTIVATE_ON_RECEIPT verb was accepted. The program specified (PGM) is activated when data or other information is available for receipt.                                                                                                    |
| LU62RC_SKCNFRM_BADSTATE | 00000032  | A state check occurred. This can occur if a CONFIRM or an ACTIVATE_ON_CONFIRMATION with VERB=CONFIRM is issued when the conversation is not in <u>send</u> state or while in the middle of sending a logical record.                            |
| LU62RC_SKCNFRM_INVALID  | 00000033  | A state check occurred. A CONFIRM or an ACTIVATE_ON_CONFIRMATION with VERB=CONFIRM was issued for a conversation allocated with a SYNC level of NONE.                                                                                           |
| LU62RC_SKCNFMD_BADSTATE | 00000041  | A state check occurred. A CONFIRMED was issued, but the conversation is not in <u>received-confirm</u> , <u>received-confirm-send</u> , or <u>received-confirm-deallocate</u> state.                                                            |
| LU62RC_PKDLLOC_BADTYPE  | 00000051  | A parameter check was detected. An ACTIVATE_ON_CONFIRMATION with VERB=DEALLOCATE or a DEALLOCATE with TYPE=CONFIRM was requested on a conversation allocated with SYNC level of NONE.                                                           |
| LU62RC_SKDLLOC_FLUSH    | 00000052  | A state check occurred. This can occur when a DEALLOCATE with TYPE=FLUSH is issued when the conversation is not in <u>send</u> state or while in the middle of sending a logical record.                                                        |
| LU62RC_SKDLLOC_CONFIRM  | 00000053  | A state check occurred. This can occur when an ACTIVATE_ON_CONFIRMATION with VERB=DEALLOCATE or a DEALLOCATE with TYPE=CONFIRM is issued when the conversation is not in <u>send</u> state, or while in the middle of sending a logical record. |
| LU62RC_SKDLLOC_ABEND    | 00000056  | A state check occurred. A DEALLOCATE with TYPE=ABEND was issued, but the conversation is in <u>end-conversation</u> state.                                                                                                                      |
| LU62RC_SKDLLOC_LOCAL    | 00000057  | A state check occurred. A DEALLOCATE with TYPE=LOCAL was issued, but the conversation is not in <u>end-conversation</u> state.                                                                                                                  |
| LU62RC_SKFLUSH_BADSTATE | 00000061  | A state check occurred. A FLUSH was issued, but the conversation is not in <u>send</u> state.                                                                                                                                                   |
| LU62RC_SKPOSTR_BADSTATE | 00000092  | A state check occurred. A POST_ON_RECEIPT was issued, but the conversation is not in <u>receive</u> state.                                                                                                                                      |

## TPPCC

Table 12. TPPCC Secondary Return Codes (continued)

| Symbolic Name             | Hex Value | Meaning                                                                                                                                                                                                                                                                               |
|---------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LU62RC_PKPTRCV_INVTYPE    | 000000A1  | A parameter check was detected. An ACTIVATE_ON_CONFIRMATION with VERB=PREPARE_TO_RECEIVE or a PREPARE_TO_RECEIVE with TYPE=CONFIRM was requested on a conversation allocated with SYNC level of NONE.                                                                                 |
| LU62RC_SKPTRCV_BADSTATE   | 000000A3  | A state check occurred. This can occur if a PREPARE_TO_RECEIVE is issued when the conversation is not in <u>send</u> state or while in the middle of sending a logical record.                                                                                                        |
| LU62RC_SKRECEV_BADSTATE   | 000000B1  | A state check occurred. This can occur if a RECEIVE or ACTIVATE_ON_RECEIPT is issued when the conversation is not in <u>send</u> or <u>receive</u> state or while in the middle of sending a logical record.                                                                          |
| LU62RC_SKRTSND_BADSTATE   | 000000E1  | A state check occurred. A REQUEST_TO_SEND was issued, but the conversation is not in <u>send</u> , <u>receive</u> , <u>received-confirm</u> , <u>received-confirm-send</u> , or <u>received-confirm-deallocate</u> state.                                                             |
| LU62RC_PKSENDD_BADLL      | 000000F1  | A parameter check was detected. This can occur if a SEND_DATA is issued without a data message block or with an invalid logical length or if there is bad file chaining in the data block.                                                                                            |
| LU62RC_PK_BAD_PROG        | 000000F2  | A parameter check was detected. An ACTIVATE_ON_CONFIRMATION or ACTIVATE_ON_RECEIPT verb was issued, but the program specified on the PGM parameter is not defined in the TPNT.                                                                                                        |
| LU62RC_SKSENDD_BADSTATE   | 000000F2  | A state check occurred. A SEND_DATA was issued, but the conversation is not in <u>send</u> state.                                                                                                                                                                                     |
| LU62RC_INVALID_MODE_NAME  | 000000F3  | An allocation error occurred. This error occurs when the mode name specified on the ALLOCATE verb is SNASVCMG. This mode name is reserved for CNOS processing.                                                                                                                        |
| LU62RC_PK_BAD_PARM        | 000000F3  | A parameter check was detected. An ACTIVATE_ON_CONFIRMATION or ACTIVATE_ON_RECEIPT verb was issued, but there is no block on the data level specified by the PARM parameter.                                                                                                          |
| LU62RC_SKSENDE_BADSTATE   | 00000111  | A state check occurred. A SEND_ERROR was issued, but the conversation is in an invalid state.                                                                                                                                                                                         |
| LU62RC_NOT_RCV_STATE      | 00000122  | A state check occurred. This can occur if a TEST with TEST=POSTED or WAIT is issued, but the conversation is not in <u>receive</u> state. This error also occurs when a TEST with TEST=RTSRCVD is issued while the conversation is not in either <u>send</u> or <u>receive</u> state. |
| LU62RC_TP_NOT_AVAIL_RETRY | 084B6031  | An allocation error occurred. The transaction program could not be started. One possible cause is the lack of resources. This is a temporary condition; retry the request.                                                                                                            |



Table 12. TPPCC Secondary Return Codes (continued)

| Symbolic Name                      | Hex Value | Meaning                                                                                                                                                                                  |
|------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | 084C0000  | An allocation error occurred. The transaction program could not be started. One possible cause is the lack of resources. This is a permanent condition; do not retry the request.        |
| LU62RC_TPN_NOT_RECOGNIZED          | 10086021  | An allocation error occurred. The transaction program name specified on the ALLOCATE verb was not recognized by the remote LU.                                                           |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | 10086032  | An allocation error occurred. The remote LU rejected the allocation request because the PIP parameter specified did not agree with the PIP value defined by the remote program.          |
| LU62RC_CONV_TYPE_MISMATCH          | 10086034  | An allocation error occurred. This indicates that the remote LU rejected the allocation request because the remote transaction program does not support the conversation type requested. |
| LU62RC_SYNLVL_NOTSUPPORT           | 10086041  | An allocation error occurred. An ALLOCATE verb was issued with a value for the SYNC option that the remote LU does not support.                                                          |

- In addition to the return code, symbolic names are provided for various values returned by the TPF/APPC verbs. Table 13 lists the defined symbolic names for the REQUEST\_TO\_SEND\_RECEIVED (RTSRCVD) indicator, the SYNC level, and the WHAT\_RECEIVED (WHATRCV) indicator.

Table 13. Miscellaneous TPPCC Symbolic Values

| Symbolic Name         | Value | Verb           | Meaning                                                                                                                                                                               |
|-----------------------|-------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LU62_SYNCLVL_NONE     | 01    | GET_ATTRIBUTES | The synchronization level of this conversation is NONE.                                                                                                                               |
| LU62_SYNCLVL_CONFIRM  | 02    | GET_ATTRIBUTES | The synchronization level of this conversation is CONFIRM.                                                                                                                            |
| LU62WR_SEND           | 02    | RECEIVE        | The conversation received the SEND indication. There is no message block on data level 0 or at the address specified by the DATA parameter.                                           |
| LU62WR_DATACOMPLETE   | 03    | RECEIVE        | The conversation received a complete logical record. There is a message block on data level 0 or at the address specified by the DATA parameter.                                      |
| LU62WR_DATAINCOMPLETE | 04    | RECEIVE        | The conversation received an incomplete logical record. There is a message block on data level 0 or at the address specified by the DATA parameter.                                   |
| LU62WR_CONFIRM        | 05    | RECEIVE        | The remote transaction program issued a CONFIRM verb. There is no message block on data level 0 or at the address specified by the DATA parameter.                                    |
| LU62WR_CONFIRMSEND    | 06    | RECEIVE        | The remote transaction program issued a PREPARE_TO_RECEIVE verb with the CONFIRM option. There is no message block on data level 0 or at the address specified by the DATA parameter. |
| LU62WR_CONFIRMDLLOC   | 07    | RECEIVE        | The remote transaction program issued a DEALLOCATE verb with the CONFIRM option. There is no message block on data level 0 or at the address specified by the DATA parameter.         |

## TPPCC

Table 13. Miscellaneous TPPCC Symbolic Values (continued)

| Symbolic Name      | Value | Verb                                          | Meaning                                              |
|--------------------|-------|-----------------------------------------------|------------------------------------------------------|
| LU62_RTSND_RCVDYES | 80    | CONFIRM<br>RECEIVE<br>SEND_DATA<br>SEND_ERROR | A REQUEST_TO_SEND was received on this conversation. |

## Programming Considerations

- You can execute this macro on any I-stream.
- The TPPCC macro generates inner macro calls based on the positional *verbname* parameter. Each of the inner macros generates the instructions to build a parameter list and then generates an ENTRC macro to the TPF/APPC verb router program, a real-time ECB-controlled program. Additional real-time ECB-controlled programs are activated based on the *verbname*, the passed parameters, and the status and conditions that may be present for the conversation. There must be at least 7 program nesting levels available.
- The TPPCC support programs save and restore the contents of EBW016–EBW103 and the EBX area of the calling ECB.
- The TPPCC support programs use various data levels of the ECB to perform their processing. If the data levels used by the support programs have blocks attached when the TPPCC macro is executed, the support programs detach the blocks from the ECB (using DETAC) and attach them back to the ECB (using ATTAC) before returning control to the calling program. Data level 15 (DF) is always used. Data levels 12 through 14 (DC through DE) are used at various times. The application can reduce the overhead inherent in the DETAC/ATTAC processing by ensuring that these data levels are available when the TPPCC macro is executed.
- When a remote LU 6.2 transaction program initiates a conversation with a TPF transaction program, TPF receives an ATTACH FMH5 record. See the *TPF ACF/SNA Data Communications Reference* for a description of the ATTACH interface.
- The 2 ACTIVATE verbs (ACTIVATE\_ON\_CONFIRMATION and ACTIVATE\_ON\_RECEIPT) are TPF-only extensions to the LU 6.2 architecture. They allow you to implement a TPF transaction program with an asynchronous wait mechanism. After issuing either of the ACTIVATE verbs, the ECB (which represents the transaction program instance) is expected to issue the TPF EXITC macro. When the wait implied with the synchronous type verb (RECEIVE, CONFIRM, PREPARE\_TO\_RECEIVE, DEALLOCATE) completes, TPF/APPC support re-creates the TPF transaction program instance by activating the specified program with a new ECB.
- See the individual verb descriptions for programming considerations relating to the supported verbs.

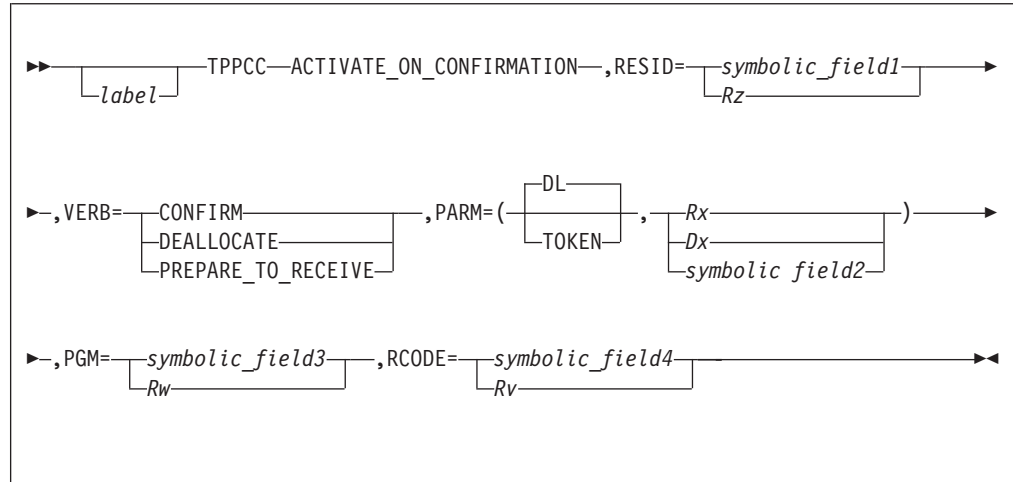
## Examples

See the individual verb sections for examples.

## TPPCC ACTIVATE\_ON\_CONFIRMATION

Use the TPPCC general macro with the ACTIVATE\_ON\_CONFIRMATION verb specified to allow the issuing TPF ECB to exit after sending out a CONFIRM, DEALLOCATE TYPE=CONFIRM, or PREPARE\_TO\_RECEIVE TYPE=CONFIRM request. TPF/APPC then activates a different ECB at the program specified when the reply to the CONFIRM request has arrived and been processed. The ACTIVATE\_ON\_CONFIRMATION verb is a TPF extension to the LU 6.2 architecture.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### ACTIVATE\_ON\_CONFIRMATION

Directs the TPPCC macro to perform the ACTIVATE\_ON\_CONFIRMATION verb function.

#### RESID

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation, or one that was assigned by an incoming ATTACH.

#### VERB

Specifies the verb function that is to be executed. The value must be one of the following:

##### CONFIRM

Specifies that the function of the CONFIRM verb should be executed.

##### DEALLOCATE

Specifies that the function of the DEALLOCATE verb with TYPE=CONFIRM should be executed.

##### PREPARE\_TO\_RECEIVE

Specifies that the function of the PREPARE\_TO\_RECEIVE verb with TYPE=CONFIRM should be executed.

## TPPCC ACTIVATE\_ON\_CONFIRMATION

### PARM

Specifies the parameter information to be passed to the new ECB. The type of information is based on the option specified, which must be one of the following:

### DL

Specifies a TPF data level within the range D0 through DA. After the reply to the CONFIRM request arrives and is processed, the contents of the working storage block on the data level specified is passed from the ECB issuing this verb to a 4K block on the same data level of the ECB created.

### TOKEN

Specifies the symbolic name of a field or a register that points to a field. This is an 8-byte field that is saved and passed to EBX000 of the new ECB that is created when the reply to the CONFIRM request has arrived and been processed.

### PGM

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field containing the TPF real-time program segment to be activated when the reply to the CONFIRM request has arrived and been processed. This TPF real-time segment must be defined in the TPF transaction program name table (TPNT) as an ACTIVATE\_ON\_RECEIPT target program. This is done with the ITPNT macro by specifying TYPE = AOR and by specifying the segment name as both the TPN name and the PGM name. See “ITPNT–Transaction Program Name Table Macro” on page 278 for more information about the ITPNT macro.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

## Entry Requirements

- The conversation must be in send state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- If the return code is LU62RC\_OK, the ECB is no longer connected to the transaction program or the conversation. The ECB cannot issue any further TPF/APPC verbs.
- If the return code is any value other than LU62RC\_OK, the ECB is still connected to the transaction program and the conversation. However, since all other return codes indicate a serious error, this also indicates that a DEALLOCATE ABEND occurred for the conversation. The ECB cannot issue any further TPF/APPC verbs for this conversation but can issue TPF/APPC verbs for other conversations associated with the transaction program instance.

**Note:** The transaction program is identified with the transaction program identifier (TCB ID), and the conversation is identified with the conversation control block identifier (CCB ID). Both identifiers are defined in the ECB as EBTCBID and EBCCBID, respectively.

- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.

## TPPCC ACTIVATE\_ON\_CONFIRMATION

- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the ACTIVATE\_ON\_CONFIRMATION verb. A complete list of return codes and their definitions can be found Table 11 on page 421 and Table 12 on page 422.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_AOC_INCOMPLETE   | ....         | 00000009       |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_PKDLLOC_BADTYPE  | ....         | 00000051       |
| LU62RC_PKPTRCV_INVTYPE  | ....         | 000000A1       |
| LU62RC_PK_BAD_PROG      | ....         | 000000F2       |
| LU62RC_PK_BAD_PARM      | ....         | 000000F3       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKCNFRM_BADSTATE | ....         | 00000032       |
| LU62RC_SKCNFRM_INVALID  | ....         | 00000033       |
| LU62RC_SKDLLOC_CONFIRM  | ....         | 00000053       |
| LU62RC_SKPTRCV_BADSTATE | ....         | 000000A3       |
| LU62RC_TPF_ABEND        | FFFF         |                |

- The following table contains a list of the primary and secondary return codes that can be passed to the program specified by PGM when it is activated. The return codes are based upon the reply to the CONFIRM request.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NO_RETRY           | 0010         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## TPPCC ACTIVATE\_ON\_CONFIRMATION

### Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- You can issue this verb instead of the CONFIRM, DEALLOCATE TYPE=CONFIRM or PREPARE\_TO\_RECEIVE TYPE=CONFIRM verb. An ECB is not suspended while waiting for the reply to the CONFIRM request.
- After the reply to the confirmation request arrives and is processed,
  - TPF/APPC support activates the program specified by the PGM parameter
  - The return codes are passed to the new ECB at EBX008
  - The transaction program identifier (TCB ID) and the conversation control block identifier (CCB ID) are passed to the new ECB in fields EBTCBID and EBCCBID, respectively.
- The association of the ECB with the transaction program instance is dependent on the return code.
- Any working storage block passed on a data level to the activated program is released by TPF/APPC.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

### Examples

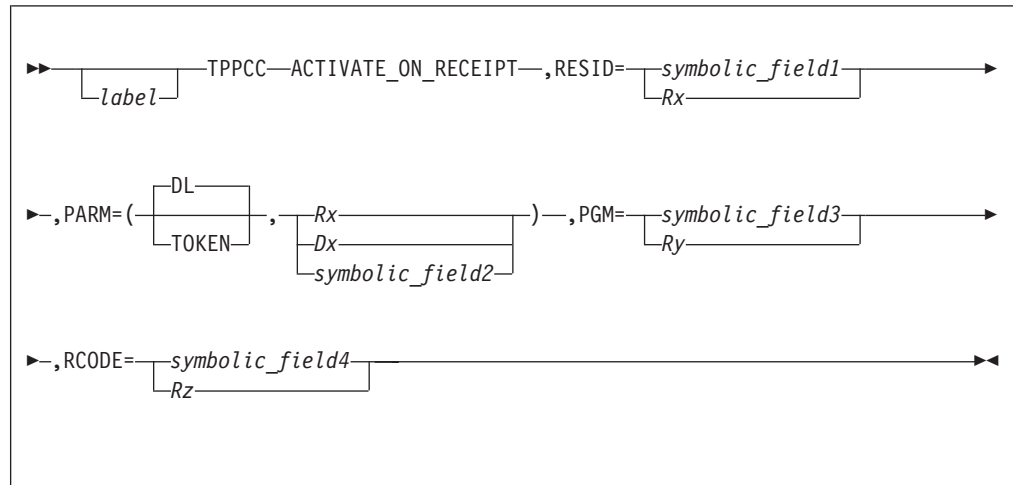
```
SYMB100 TPPCC ACTIVATE_ON_CONFIRMATION, X
 RESID=(R7), X
 VERB=CONFIRM, X
 PARM=(DL,D5), X
 PGM=(R2), X
 RCODE=EBW010 X

SYMB100 TPPCC ACTIVATE_ON_CONFIRMATION, X
 RESID=(R7), X
 VERB=PREPARE_TO_RECEIVE, X
 PARM=(TOKEN,(R6)), X
 PGM=(R2), X
 RCODE=EBW010
```

## TPPCC ACTIVATE\_ON\_RECEIPT

Use the TPPCC general macro with the ACTIVATE\_ON\_RECEIPT verb specified to allow the issuing TPF ECB to exit and activate a different ECB at the program specified after information has been received. The information received may be data, conversation status, or a confirmation request. The ACTIVATE\_ON\_RECEIPT verb is a TPF extension to the LU 6.2 architecture.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **ACTIVATE\_ON\_RECEIPT**

Directs the TPPCC macro to perform the ACTIVATE\_ON\_RECEIPT verb function.

#### **RESID**

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### **PARM**

Specifies the parameter information to be passed to the new ECB. The type of information is based on the option specified, which must be one of the following:

##### **DL**

Specifies a TPF data level within the range D1 through DA. After the information is received, the contents of the working storage block on the data level specified is passed from the ECB issuing this verb to a 4K block on the same data level of the ECB created.

##### **TOKEN**

Specifies the symbolic name of a field or a register that points to a field. This is an 8-byte field that is saved and passed to EBX000 of the new ECB that is created after the information is received.

##### **PGM**

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field containing the TPF real-time program segment to be activated after the information has been received. This TPF real-time segment must be

## TPPCC ACTIVATE\_ON\_RECEIPT

defined in the TPF transaction program name table (TPNT) as an ACTIVATE\_ON\_RECEIPT target program. This is done with the ITPNT macro by specifying TYPE=AOR and by specifying the segment name as both the TPN name and PGM name. See “ITPNT–Transaction Program Name Table Macro” on page 278 for more information about the ITPNT macro.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

## Entry Requirements

- The conversation must be in send or receive state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- If the return code is LU62RC\_OK, the ECB is no longer connected to the transaction program or the conversation. The ECB cannot issue any further TPF/APPC verbs.
- If the return code is any value other than LU62RC\_OK, the ECB is still connected to the transaction program and the conversation. However, since all other return codes indicate a serious error, this also indicates that a DEALLOCATE ABEND occurred for the conversation specified on the ACTIVATE\_ON\_RECEIPT verb. The ECB cannot issue any further TPF/APPC verbs for this conversation but can issue TPF/APPC verbs for other conversations associated with the transaction program instance.

**Note:** The transaction program is identified with the transaction program identifier (TCB ID), and the conversation is identified with the conversation control block identifier (CCB ID). Both identifiers are defined in the ECB as EBTCBID and EBCCBID, respectively.

- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the ACTIVATE\_ON\_RECEIPT verb. A complete list of return codes and their definitions can be found Table 11 on page 421 and Table 12 on page 422.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_AOR_INCOMPLETE   | ....         | 00000009       |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_PK_BAD_PROG      | ....         | 000000F2       |
| LU62RC_PK_BAD_PARM      | ....         | 000000F3       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKRECEV_BADSTATE | ....         | 000000B1       |
| LU62RC_TPF_ABEND        | FFFF         |                |



- The following table contains a list of the primary and secondary return codes that can be passed to the program specified by PGM when it is activated. The return codes are based upon the processing of the RECEIVE function.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_DLLOC_NORMAL                | 0009         |                |
| LU62RC_PGMERR_NOTRUNC              | 000C         |                |
| LU62RC_PGMERR_TRUNC                | 000D         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NO_RETRY           | 0010         |                |
| LU62RC_SVCERR_NOTRUNC              | 0011         |                |
| LU62RC_SVCERR_TRUNC                | 0012         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- You can issue this verb instead of the RECEIVE verb. If there is no information available to satisfy a RECEIVE when the ACTIVATE\_ON\_RECEIPT verb is issued, an ECB is not suspended waiting for the information to arrive. When information arrives, TPF/APPC support creates a new ECB, performs a RECEIVE, and activates the program specified by PGM. The information that was received is passed to the activated program.
- When the program specified by PGM is activated, this indicates that a RECEIVE was done. Check the return codes passed to the program at EBX008 to determine the success of the RECEIVE. If the return code is LU62RC\_OK, check the WHAT\_RECEIVED field (passed at EBX014) to see what was received. If WHAT\_RECEIVED is DATA (COMPLETE or INCOMPLETE), the data is passed to the program on D0. The transaction program identifier (TCB ID) and the conversation control block identifier (CCB ID) are passed to the new ECB in fields EBTCBID and EBCCBID, respectively.

## TPPCC ACTIVATE\_ON\_RECEIPT

- If there is information available when the ACTIVATE\_ON\_RECEIPT verb is issued, TPF/APPC activates the program specified by PGM right away with a new ECB. The information is not returned to the program that issues the ACTIVATE\_ON\_RECEIPT verb.
- The association of the ECB with the transaction program instance is dependent on the return code.
- Any working storage block passed on a data level to the activated program is released by TPF/APPC.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

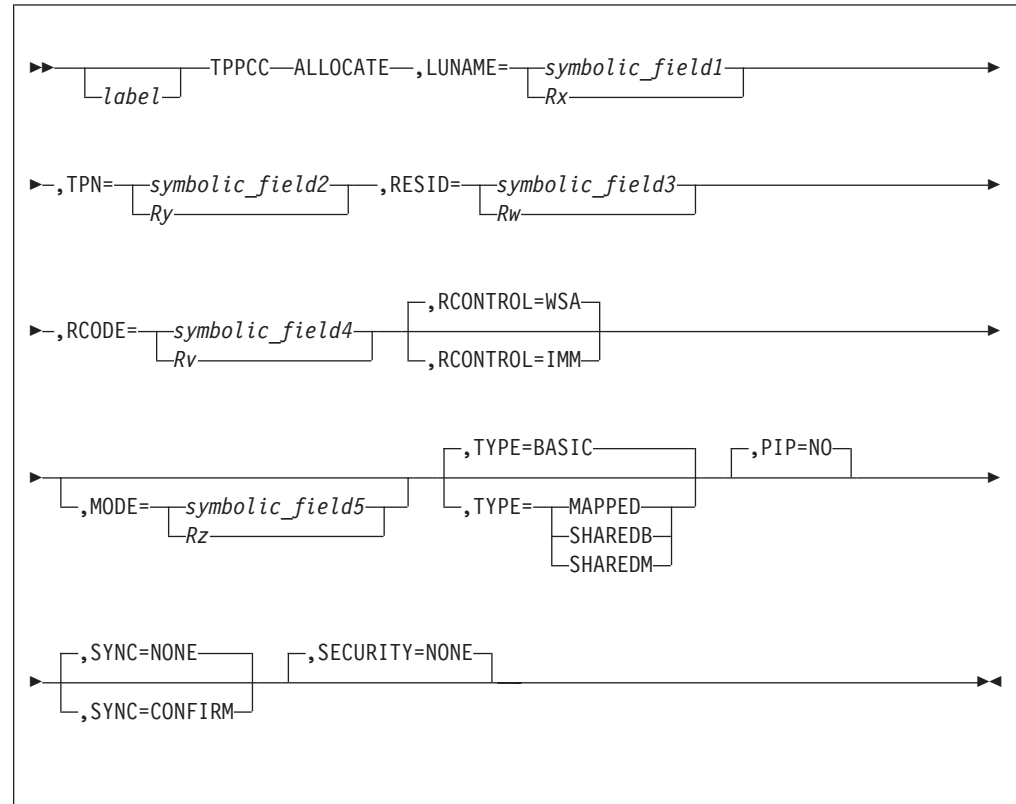
```
SYMB100 TPPCC ACTIVATE_ON_RECEIPT, X
 RESID=(R7), X
 PARM=(DL,D5), X
 PGM=(R2), X
 RCODE=EBW010

SYMB100 TPPCC ACTIVATE_ON_RECEIPT, X
 RESID=(R7), X
 PARM=(TOKEN,(R6)), X
 PGM=(R2), X
 RCODE=EBW010
```

# TPPCC ALLOCATE

Use the TPPCC general macro with the ALLOCATE verb specified to allocate a conversation between a TPF transaction program and a transaction program in a remote LU. A resource ID is assigned to the conversation and returned by this macro.

## Format



### label

A symbolic name can be assigned to the macro statement.

### ALLOCATE

Directs the TPPCC macro to perform the ALLOCATE verb function.

### LUNAME

Specifies the symbolic name of a field or a register pointing to a field. This is a 16-byte field that contains the network name of the remote LU or local secondary LU (SLU) thread with which this local transaction program wants to start a conversation. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

### TPN

Specifies either the symbolic name of a field or a register pointing to a field that contains the following:

- The 1-byte length of the remote transaction program name.
- A remote transaction program name, which can be from 4 to 64 characters long.

### RCONTROL

Specifies when control is returned to the issuer. The allowed values are:

## TPPCC ALLOCATE

### WSA

Specifies that control is returned when a session is allocated for this conversation.

### IMM

Specifies to allocate a session for the conversation if a session is immediately available. A session is immediately available when it is active, it is not allocated to another conversation, and the local LU is the contention winner for the session.

**Note:** TPF does not support the other options defined for this parameter by the LU 6.2 architecture.

### MODE

Specifies the name used to designate the properties of the session to be allocated. It can be specified as a symbolic name of a field or a register, which points to a field that contains the 8-character mode name (padded on the right with blanks if necessary).

**Note:** If you do not specify MODE, it defaults to the single session mode name. The single session mode name is defined with the SINGMODE parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for more information about the SNAKEY macro.

### TYPE

Specifies the type of conversation. The allowed values are:

#### BASIC

Provides support for the BASIC\_CONVERSATION option defined by the LU 6.2 architecture. This is the default.

#### MAPPED

Provides support for the MAPPED\_CONVERSATION option defined by the LU 6.2 architecture.

**Note:** TPF/APPC supports mapped conversations through the C language only. See *TPF C/C++ Language Support User's Guide* for information on the mapped conversation verbs.

#### SHAREDDB

Provides support for the BASIC\_CONVERSATION option defined by the LU 6.2 architecture, and specifies that this is a shared LU 6.2 conversation.

#### SHAREDMM

Provides support for the MAPPED\_CONVERSATION option defined by the LU 6.2 architecture, and specifies that this is a shared LU 6.2 conversation.

**Note:** TPF/APPC supports mapped conversations through the C language only. See *TPF C/C++ Language Support User's Guide* for information on the mapped conversation verbs.

### PIP

NO is the only allowable value. PIP (program initialization parameter) data cannot be supplied by the TPF transaction program. If PIP data is indicated in an ATTACH header received by TPF, the ATTACH conversation request is rejected.

### SYNC

Specifies the synchronization level allowed on this conversation. The allowed values are:

**NONE**

Specifies that the programs cannot perform confirmation processing on this conversation. This is the default value.

**CONFIRM**

Specifies that the programs can perform confirmation processing on this conversation.

**Note:** TPF does not support the LU 6.2 architecture's SYNCPT option.

**SECURITY**

Specifies the security level allowed on this conversation. NONE is the only allowed value. (For migration purposes, NONE can be abbreviated as NO.) TPF does not support the options SAME and PGM defined by the LU 6.2 architecture.

**RESID**

Specifies either the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is returned. This resource ID must be specified on all subsequent verbs for this conversation.

**RCODE**

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

## Entry Requirements

See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the ALLOCATE verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name            | Primary Code | Secondary Code |
|--------------------------|--------------|----------------|
| LU62RC_OK                | 0000         |                |
| LU62RC_ALLOC_ERROR       | 0003         |                |
| LU62RC_ALLOCERR_NORETRY  | ....         | 00000004       |
| LU62RC_ALLOCERR_RETRY    | ....         | 00000005       |
| LU62RC_INVALID_MODE_NAME | ....         | 000000F3       |
| LU62RC_ALLOC_UNSUCESSFUL | 0015         |                |
| LU62RC_TPF_ABEND         | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.

## TPPCC ALLOCATE

- The value returned in RESID must be used by all other TPF/APPC verbs issued for this conversation.
- If the TYPE parameter is SHAREDDB or SHAREDLM, then this is a shared LU 6.2 conversation. Shared LU 6.2 conversations are one-way pipes used to send data from the TPF system to a remote LU. Verbs can be issued from any ECB for a shared conversation. For conversations that are not shared, verbs can be issued only from the ECB that started the conversation. Because a shared conversation is a one-way pipe used to send data, only certain LU 6.2 verbs are allowed:
  - SEND\_DATA
  - FLUSH
  - GET\_ATTRIBUTES
  - DEALLOCATE (except when TYPE=CONFIRM is specified)

See *TPF ACF/SNA Data Communications Reference* for more information about shared LU 6.2 conversations.

- The value returned in RESID is stored in field EBCCBID in the ECB. This changes with every ALLOCATE request.
- If you are allocating a secondary LU (SLU) thread session, you must specify a single session mode name. Parallel sessions are not supported for SLU threads.
- The ATTACH is buffered until the buffer is full or a verb that implies the FLUSH function is issued.
- If the session assigned to this conversation is a contention winner, the session is assigned without seeking permission from the remote partner. If the RCONTROL parameter is WSA and if the session assigned to this conversation is a contention loser, permission is sought from the remote partner before the allocation is permitted.
- A session is activated for this conversation if all of the following conditions are true:
  1. The RCONTROL parameter is WSA
  2. A session is not already available for the conversation
  3. The session limit has **not** been reached for this LUNAME and MODE.

**Note:** PU 2.1 SLU thread sessions cannot be activated from the TPF side. In this case, if you specify a local SLU thread, that SLU must already be in session with the remote LU.

This verb uses TPF's EVENT and POST facility to suspend the ECB until a session is established. If a session is not established within a certain amount of time, the program sends a failure return code to the transaction program. The amount of time that the system will wait is determined by the value you specify for the TPALLOC parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.

ALLOCATE must know what 2 LUs are involved to activate the session. One LU is specified with LUNAME; the other LU is determined as follows:

- If you specify a remote LU that is in session with SLU threads, ALLOCATE selects the thread to be used for the session.
- If you specify a local LU that is a SLU thread, a remote LU must have been previously initialized for that SLU thread (with a CNOS INITIALIZE request).
- If you specify a parallel sessions mode name, the LU specified on the CNOS INITIALIZE request is used.
- The default TPF/APPC LU is used if the following conditions are true:
  - You specify single sessions

- You are not using SLU threads
- A CNOS INITIALIZE was not done.
- The LU specified on the CNOS INITIALIZE is used if the following conditions are true:
  - You specify single sessions
  - A CNOS INITIALIZE was done.
- The ALLOCATE request is queued until a session becomes available for use by this conversation if all of the following conditions are true:
  1. The RCONTROL parameter is WSA
  2. A session is not already available for this conversation
  3. The session limits have been reached for this LUNAME and MODE.
- If the conversation is allocated to a contention loser session, this verb uses TPF's EVENT and POST facility to suspend the ECB until the program receives a BID response.
 

If the program does not receive a BID response from the remote LU within a certain amount of time, an UNBIND is scheduled for this session, and the program sends a failure return code to the transaction program. The amount of time that the system waits is determined by the value you specify for the TPALLOC parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- If the ECB that issues the ALLOCATE request does not already have a TCB ID stored in field EBTCBID in the ECB, the ALLOCATE request represents a new TPF transaction program instance, and a new TCB ID is stored in EBTCBID. If EBTCBID does already have a value stored, the ALLOCATE request represents another conversation for the same transaction program instance, and EBTCBID is not changed.
- The mode name SNASVCMG cannot be used by a user TPF TP.
- Upon completion of the ALLOCATE, the local transaction program is in send state, and the remote partner is in receive state.
- See "Programming Considerations" on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

```

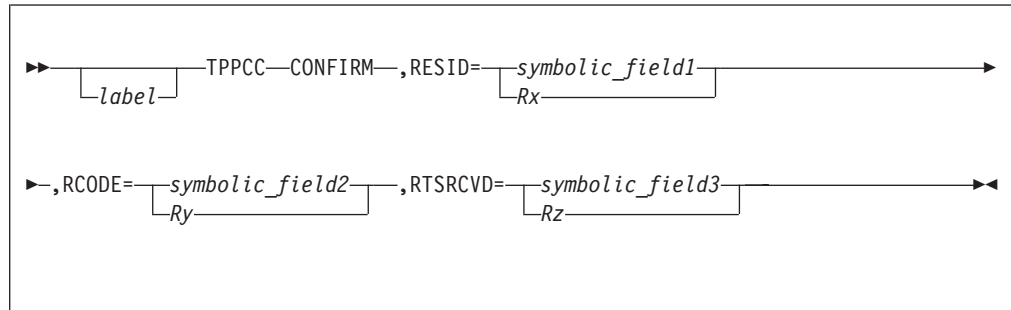
SYMB100 TPPCC ALLOCATE, X
 LUNAME=EBX024, X
 TPN=EBX000, X
 RCONTROL=WSA, X
 MODE=(R3) X
 TYPE=BASIC, X
 PIP=NO, X
 SYNC=NONE, X
 SECURITY=NONE, X
 RESID=(R7), X
 RCODE=EBW010

```

## TPPCC CONFIRM

Use the TPPCC general macro with the CONFIRM verb specified to send a confirmation request to the remote transaction program and wait for the confirmation reply. This allows the 2 programs to synchronize their processing.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### CONFIRM

Directs the TPPCC macro to perform the CONFIRM verb function.

#### RESID

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

#### RTSRCVD

Specifies the symbolic name of a 1-byte field or a register that contains a value that indicates whether REQUEST\_TO\_SEND has been received. If a REQUEST\_TO\_SEND has been received, the value is LU62\_RTSND\_RCVDYES.

### Entry Requirements

- The conversation must be in send state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

### Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the CONFIRM verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in



Table 12 on page 422.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKCNFRM_BADSTATE            | ....         | 00000032       |
| LU62RC_SKCNFRM_INVALID             | ....         | 00000033       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- This verb uses TPF's EVENT and POST facility to suspend the ECB until the program receives a confirmation reply. Since the ECB is suspended, all unnecessary resources should be released before this verb is issued. Failure to do so can cause serious system performance degradation.

**Note:** You can avoid the problem of suspended ECBs by using the ACTIVATE\_ON\_CONFIRMATION verb instead of CONFIRM. See “TPPCC ACTIVATE\_ON\_CONFIRMATION” on page 427 for more information.

- If the program does not receive a confirmation reply within a certain amount of time, the TPF/APPC support issues a DEALLOCATE TYPE=ABEND to terminate the conversation. The amount of time that the system waits is determined by the value you specify for the TPRECV parameter on the SNAKEY macro. See the *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- When the value of the RTSRCVD parameter is LU62\_RTSND\_RCVDYES, the remote program is requesting the local TPF transaction program to enter receive

## TPPCC CONFIRM

state and thereby place the remote program in send state. The local TPF transaction program enters receive state by issuing a RECEIVE verb or a PREPARE\_TO\_RECEIVE verb. The remote partner program enters the corresponding send state when it issues a RECEIVE verb and receives the SEND indicator on the WHATRCV parameter.

- If RCODE is LU62RC\_PGMERR\_PURGING or LU62RC\_SVCERR\_PURGING, the conversation enters receive state.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

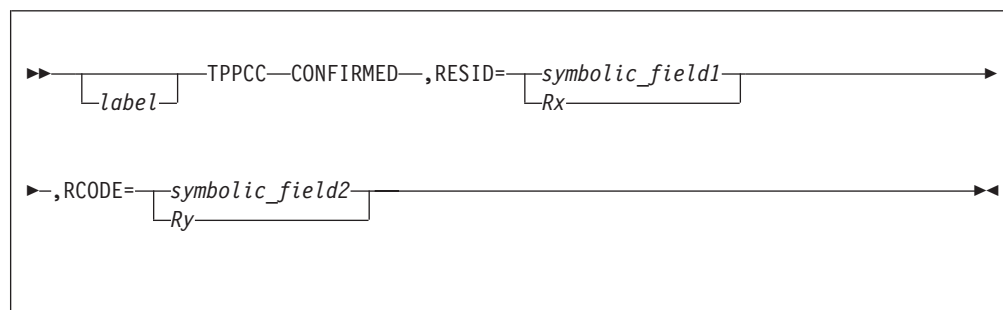
## Examples

|         |                |   |
|---------|----------------|---|
| SYMB100 | TPPCC CONFIRM, | X |
|         | RESID=EBW004,  | X |
|         | RCODE=EBW010,  | X |
|         | RTSRCVD=EBW016 |   |

## TPPCC CONFIRMED

Use the TPPCC general macro with the CONFIRMED verb specified to send a confirmation reply to the remote transaction program. This allows the 2 programs to synchronize their processing. The local TPF transaction program can issue this verb when it receives a confirmation request from the remote transaction program. See the WHATRCV parameter of the RECEIVE verb (see “TPPCC RECEIVE” on page 462).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **CONFIRMED**

Directs the TPPCC macro to perform the CONFIRMED verb function.

#### **RESID**

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### **RCODE**

Specifies the symbolic name of a field or a register that points to a field. This field is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

### Entry Requirements

- The conversation must be in one of the following states:
  - received-confirm
  - received-confirm-send
  - received-confirm-deallocate
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

### Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the CONFIRMED verb. A complete list of return codes and their definitions can be found in Table 11 on page 421

## TPPCC CONFIRMED

and in Table 12 on page 422.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKCNFMD_BADSTATE | ....         | 00000041       |
| LU62RC_TPF_ABEND        | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- A transaction program can issue this verb only as a reply to a confirmation request; the verb cannot be issued at any other time. Transaction programs can use this verb for various application-level functions. For example, the remote program can send data followed by a confirmation request. When the local program receives the confirmation request, it can issue this verb as an indication that it received and processed the data without error.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

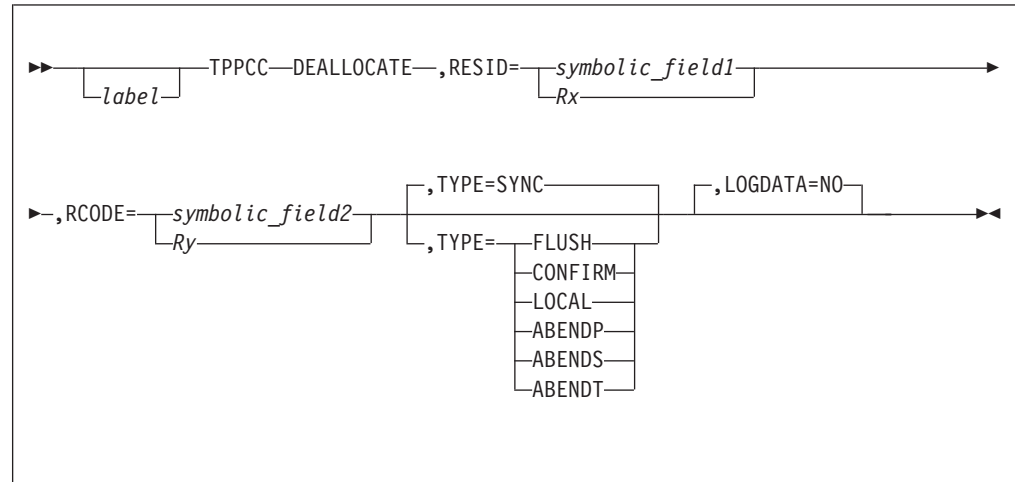
## Examples

```
SYMB100 TPPCC CONFIRMED, X
 RESID=(R4), X
 RCODE=EBW010
```

## TPPCC DEALLOCATE

Use the TPPCC general macro with the DEALLOCATE verb specified to deallocate the specified conversation from the transaction program. An implied FLUSH is executed and the resource ID becomes unassigned when the deallocation is complete.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### DEALLOCATE

Directs the TPPCC macro to perform the DEALLOCATE verb function.

#### RESID

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### TYPE

Specifies the type of deallocation to be performed. The value must be one of the following:

##### SYNC

Specifies that either the FLUSH or CONFIRM function should be performed before the conversation is deallocated, depending on the SYNC level specified at ALLOCATE time.

##### FLUSH

Specifies that the function of the FLUSH verb should be executed, and then the conversation should be deallocated.

##### CONFIRM

Specifies that the function of the CONFIRM verb should be executed, and then the conversation should be deallocated.

##### LOCAL

Specifies that the conversation should be deallocated locally. This type of deallocation can be specified **only** if the conversation is already in end-conversation state.

## TPPCC DEALLOCATE

### ABENDP|ABENDS|ABENDT

The 3 abend parameters specify that the conversation should be unconditionally deallocated. Logical record truncation can occur when the conversation is in send state, and data purging can occur when the conversation is in receive state. The only difference among the 3 abend codes is the sense codes used to notify the remote transaction program. ABENDP is intended to be used by the application transaction program to indicate that it is requesting the deallocation. ABENDS and ABENDT are intended to be used by the TPF/APPC support routines when they request the deallocation.

### LOGDATA

Specifies whether error information should be logged. NO is the only option supported. TPF does not support the YES option defined by the LU 6.2 architecture.

### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

## Entry Requirements

- The conversation must be in the following state, depending on the type of deallocation requested:

| Type    | State Required                                                                                                                |
|---------|-------------------------------------------------------------------------------------------------------------------------------|
| FLUSH   | <u>send</u>                                                                                                                   |
| CONFIRM | <u>send</u>                                                                                                                   |
| LOCAL   | <u>end-conversation</u>                                                                                                       |
| ABEND   | <u>send</u> , <u>receive</u> , <u>received-confirm</u> , <u>received-confirm-send</u> , or <u>received-confirm-deallocate</u> |

- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the DEALLOCATE verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name          | Primary Code | Secondary Code |
|------------------------|--------------|----------------|
| LU62RC_OK              | 0000         |                |
| LU62RC_PARAMETER_CHECK | 0001         |                |
| LU62RC_PK_BAD_TCBID    | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID   | ....         | 00000002       |
| LU62RC_PKDLLOC_BADTYPE | ....         | 00000051       |

## TPPCC DEALLOCATE

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKDLLOC_FLUSH               | ....         | 00000052       |
| LU62RC_SKDLLOC_CONFIRM             | ....         | 00000053       |
| LU62RC_SKDLLOC_ABEND               | ....         | 00000056       |
| LU62RC_SKDLLOC_LOCAL               | ....         | 00000057       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- When the DEALLOCATE is complete, the conversation identified by the resource ID is completed, and no further verbs can be issued for that conversation.
- If the transaction program exits without deallocating a conversation, TPF does not deallocate that conversation. The program should call this macro for all conversations before exiting.
- If TYPE=CONFIRM is specified, the normal response from the remote transaction program is CONFIRMED. However, if the remote issues a DEALLOCATE TYPE=ABEND, the local transaction program goes into end conversation state. If the remote transaction program issues a SEND\_ERROR, the local program goes into receive state, and the conversation continues.
- If TYPE=CONFIRM is specified, TPF's EVENT and POST facility is used to suspend the ECB until the program receives the confirmation reply. Since the ECB is suspended, all unnecessary resources should be released before this verb is issued. Failure to do so can cause serious system degradation.

**Note:** You can avoid the problem of suspended ECBs by using the ACTIVATE\_ON\_CONFIRMATION verb instead of DEALLOCATE. See "TPPCC ACTIVATE\_ON\_CONFIRMATION" on page 427 for more information.

## TPPCC DEALLOCATE

- If TYPE=CONFIRM is specified and the program does not receive a confirmation reply within a certain amount of time, the TPF/APPC support issues an unbind to terminate the conversation. The amount of time that the system waits is determined by the value you specify for the TPRECV parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- The remote transaction program receives the deallocation notification by means of either a return code or the WHAT\_RECEIVED indication. (See the WHATRCV parameter of the RECEIVE verb; “TPPCC RECEIVE” on page 462).
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

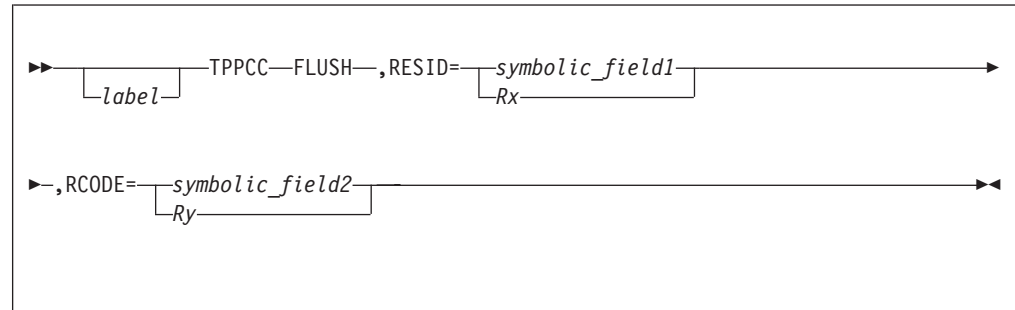
```
SYMB100 TPPCC DEALLOCATE, X
 RESID=EBW004, X
 TYPE=FLUSH, X
 LOGDATA=NO, X
 RCODE=EBW010
```



## TPPCC FLUSH

Use the TPPCC general macro with the FLUSH verb specified to allow the application to transmit any data from the local LU's data buffer, including an ATTACH from the ALLOCATE verb.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **FLUSH**

Directs the TPPCC macro to perform the FLUSH verb function.

#### **RESID**

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### **RCODE**

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is returned. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. the actual values returned are listed in “Return Conditions”.

## Entry Requirements

- The conversation must be in send state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the FLUSH verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name          | Primary Code | Secondary Code |
|------------------------|--------------|----------------|
| LU62RC_OK              | 0000         |                |
| LU62RC_PARAMETER_CHECK | 0001         |                |

## TPPCC FLUSH

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKFLUSH_BADSTATE | ....         | 00000061       |
| LU62RC_TPF_ABEND        | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- This verb is useful for optimization of processing between the local and remote transaction programs. The TPF/APPC support code buffers the information from consecutive SEND\_DATA verbs until it has a sufficient amount of information for transmission. At that time, it transmits the buffered data. However, the local TPF transaction program can issue the FLUSH verb in order to cause the TPF/APPC support code to transmit any buffered data. In this way, the local program can minimize the delay in the remote transaction program's processing. The TPF/APPC support code actually flushes the buffered data only when it has some data buffered; if there is no data buffered, nothing is transmitted to the remote LU. The buffer size is determined by the maximum request unit (RU) size of the session, which is negotiated at BIND time.
- See "Programming Considerations" on page 426 for the programming considerations relating to the TPPCC macro in general.

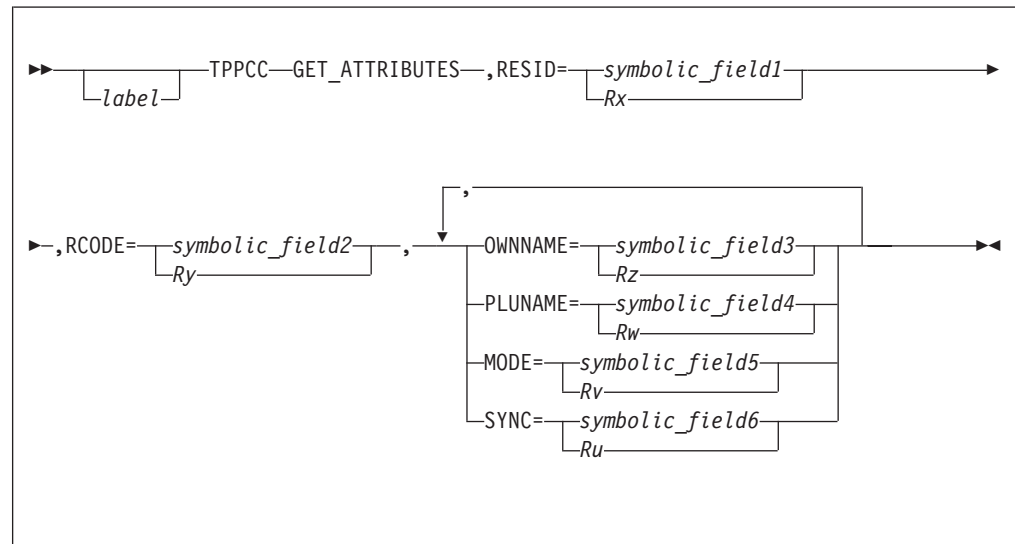
## Examples

```
SYMB100 TPPCC FLUSH, X
 RESID=EBW004, X
 RCODE=EBW010
```

## TPPCC GET\_ATTRIBUTES

Use the TPPCC general macro with the GET\_ATTRIBUTES verb specified to return information pertaining to a conversation.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### GET\_ATTRIBUTES

Directs the TPPCC macro to perform the GET\_ATTRIBUTES verb function.

#### RESID

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions" on page 452.

#### OWNAME

Specifies the symbolic name of a field or a register that points to a field. The field is a 16-byte field in which the network name of the local TPF LU is returned. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified LU name, which is padded with blanks.

#### PLUNAME

Specifies the symbolic name of a field or a register that points to a field. This is a 16-byte field in which the network name of the partner (remote) LU is returned. The first 8 bytes contain the left-justified network name, which is padded with blanks, or all blanks if the LU name is unqualified. The second 8 bytes contain the left-justified partner LU name, which is padded with blanks. This returned LU name is the name of the LU in which the remote transaction program is located.

## TPPCC GET\_ATTRIBUTES

### MODE

Specifies the symbolic name of a field or a register that points to a field. This is an 8-byte field in which the mode name of the conversation is returned.

### SYNC

Specifies the symbolic name of a field or a register that points to a field. This is a 1-byte field in which a value is returned that indicates the synchronization level of this conversation. The possible values are:

#### LU62\_SYNCLVL\_NONE

Specifies synchronization is not allowed on this conversation.

#### LU62\_SYNCLVL\_CONFIRM

Specifies CONFIRM synchronization is allowed on this conversation.

TPF does not support the SYNCPT level defined by the LU 6.2 architecture.

## Entry Requirements

- The conversation can be in any state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the GET\_ATTRIBUTES verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name          | Primary Code | Secondary Code |
|------------------------|--------------|----------------|
| LU62RC_OK              | 0000         |                |
| LU62RC_PARAMETER_CHECK | 0001         |                |
| LU62RC_PK_BAD_TCBID    | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID   | ....         | 00000002       |
| LU62RC_TPF_ABEND       | FFFF         |                |

## Programming Considerations

- TPF does not support the other parameters defined by the LU 6.2 architecture for the GET\_ATTRIBUTES verb.
- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- You must specify at least one of the parameters OWNAME, PLUNAME, MODE, or SYNC.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

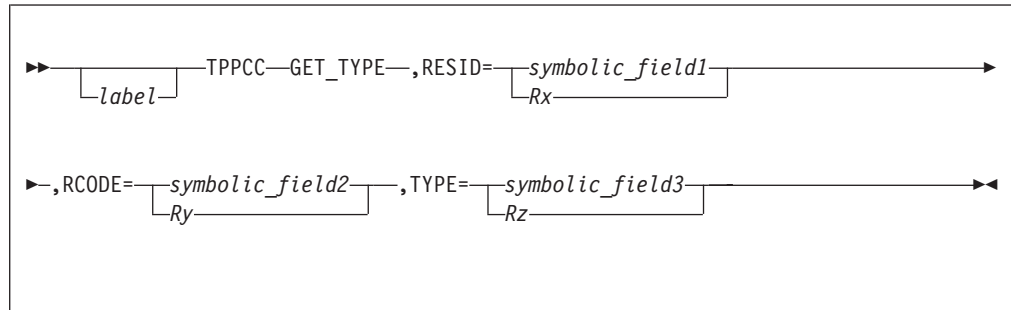
Examples

|         |                       |   |
|---------|-----------------------|---|
| SYMB100 | TPPCC GET_ATTRIBUTES, | X |
|         | RESID=EBW004,         | X |
|         | RCODE=EBW064,         | X |
|         | OWNAME=(R4),          | X |
|         | PLUNAME=EBW036,       | X |
|         | MODE=EBW058,          | X |
|         | SYNC=EBW060           |   |

## TPPCC GET\_TYPE

Use the TPPCC general macro with the GET\_TYPE verb specified to return the type of conversation, BASIC or MAPPED, to which the specified resource ID is assigned.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### GET\_TYPE

Directs the TPPCC macro to perform the GET\_TYPE verb function.

#### RESID

Specifies the symbolic name of a field or a register that points to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### RCODE

Specifies the symbolic name of a field or a register that points to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions".

#### TYPE

Specifies the symbolic name of a field or a register that points to a field. This is a 1-byte field in which a value is returned that indicates the type of resource allocated. The possible values are:

##### LU62\_BASIC\_CONVERSATION

Specifies a basic conversation.

##### LU62\_MAPPED\_CONVERSATION

Specifies a mapped conversation.

### Entry Requirements

- The conversation can be in any state.
- See "Entry Requirements" on page 421 for the entry requirements relating to the TPPCC macro in general.

### Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, "Return Codes".

- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the GET\_TYPE verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name          | Primary Code | Secondary Code |
|------------------------|--------------|----------------|
| LU62RC_OK              | 0000         |                |
| LU62RC_PARAMETER_CHECK | 0001         |                |
| LU62RC_PK_BAD_TCBID    | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID   | ....         | 00000002       |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

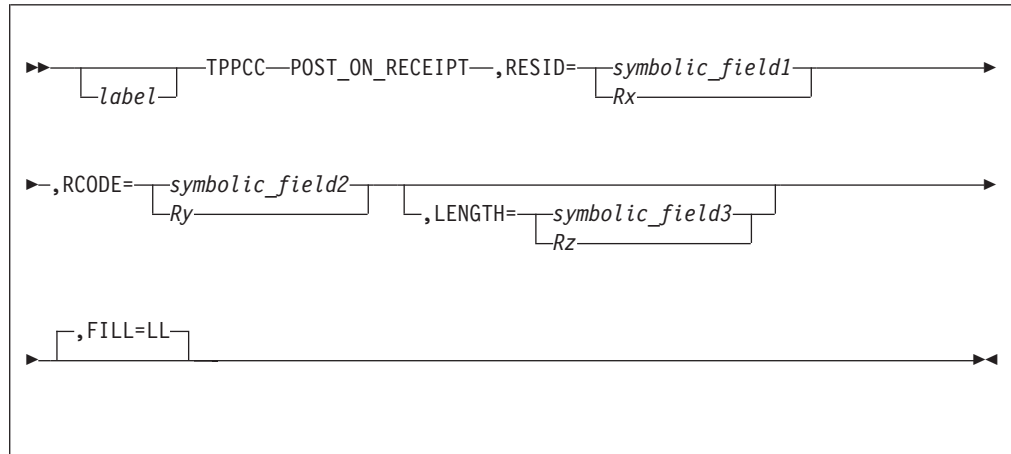
## Examples

```
SYMB100 TPPCC GET_TYPE, X
 RESID=EBW004, X
 RCODE=EBW064, X
 TYPE=(R4)
```

## TPPCC POST\_ON\_RECEIPT

Use the TPPCC general macro with the POST\_ON\_RECEIPT verb specified to cause the conversation to be posted when available information can be received by the transaction program. The information can be data or conversation information. The WAIT verb should be issued after this verb in order to wait for posting to occur. Or TEST can be issued after this verb in order to determine when posting has occurred.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### POST\_ON\_RECEIPT

Directs the TPPCC macro to perform the POST\_ON\_RECEIPT verb function.

#### RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. The resource ID must be the resource ID assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### FILL

LL specifies that posting occurs when a complete or truncated logical record is received, or when a part of a logical record is received that is at least equal in length to that specified on the LENGTH parameter. TPF does not support the BUFFER option defined by the LU 6.2 architecture.

#### LENGTH

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field in which the minimum amount of data the program must receive before the conversation can be posted is supplied. Use this parameter along with the FILL parameter to determine when to post the conversation for the receipt of data. If you omit this parameter, the conversation is posted when a complete logical record is received or other, nondata information is received (such as CONFIRM or PREPARE\_TO\_RECEIVE).

#### RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions" on page 457.



## Entry Requirements

- The conversation must be in receive state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the POST\_ON\_RECEIPT verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_INVALID_LENGTH   | ....         | 00000006       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKPOSTR_BADSTATE | ....         | 00000092       |
| LU62RC_TPF_ABEND        | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- This verb is intended for use in conjunction with the WAIT or TEST verb. Use this verb followed by a WAIT verb to allow a transaction program to perform synchronous receiving from multiple conversations. For each conversation, the program issues this verb and then issues the WAIT verb. This causes the conversations to wait until information is available to be received.  
Use this verb followed by a TEST verb to allow a transaction program to continue its processing and test the conversation to determine when information is available to be received.
- Posting becomes active for a conversation when POST\_ON\_RECEIPT is issued for a conversation. Posting is reset when one of the following verbs is issued for a conversation **after** the conversation is posted, that is, the conversation is no longer posted, but posting is still active.
  - DEALLOCATE
  - RECEIVE
  - SEND\_ERROR
  - TEST
  - WAIT.

## TPPCC POST\_ON\_RECEIPT

Posting is canceled when one of the following verbs is issued for a conversation **before** the conversation is posted, that is, posting is no longer active.

- DEALLOCATE
- SEND\_ERROR.
- Posting occurs when a complete logical record is available for the conversation, when the amount of data specified on the LENGTH parameter is available, or when information other than data is received, such as a confirmation request or an error indication. See the WHAT\_RECEIVED (WHATRCV) parameter of the RECEIVE verb for more information. (See “TPPCC RECEIVE” on page 462.)
- If the LENGTH parameter is used, the maximum length that can be entered is 32,767.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

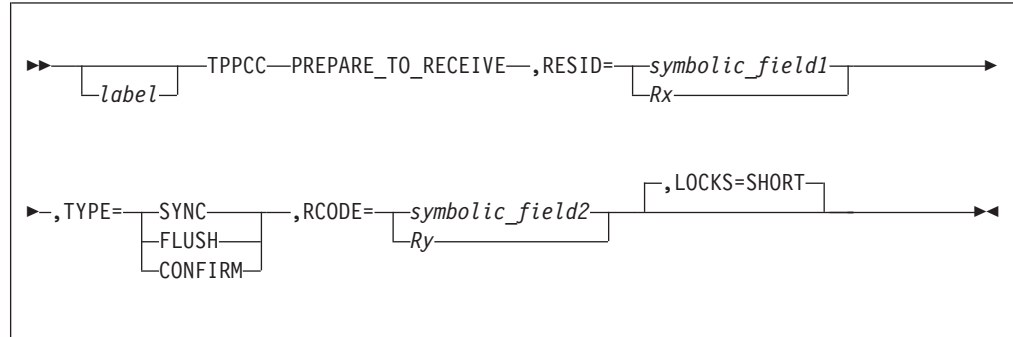
## Examples

|         |                        |   |
|---------|------------------------|---|
| SYMB100 | TPPCC POST_ON_RECEIPT, | X |
|         | RESID=(R6),            | X |
|         | FILL=LL,               | X |
|         | LENGTH=EBW040,         | X |
|         | RCODE=EBW064           |   |

## TPPCC PREPARE\_TO\_RECEIVE

Use the TPPCC general macro with the PREPARE\_TO\_RECEIVE verb specified to change the conversation from send to receive state in preparation to receive data. The change to receive state is completed as part of this verb. The execution of this verb includes the function of the FLUSH or CONFIRM verb.

### Format



#### label

A symbolic name can be assigned to the macro statement.

### PREPARE\_TO\_RECEIVE

Directs the TPPCC macro to perform the PREPARE\_TO\_RECEIVE verb function.

#### RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### TYPE

Specifies the type of PREPARE\_TO\_RECEIVE to be done on this conversation. The allowed values are:

##### CONFIRM

Performs the function of the CONFIRM verb. When the PREPARE\_TO\_RECEIVE request completes successfully, the conversation enters receive state.

##### FLUSH

Performs the function of the FLUSH verb. When the PREPARE\_TO\_RECEIVE request completes successfully, the conversation enters receive state.

##### SYNC

Performs the function of the FLUSH verb or the CONFIRM verb based on the synchronization level of the conversation. If the synchronization level is NONE, a FLUSH is performed. If the synchronization level is CONFIRM, a CONFIRM is performed.

#### LOCKS

SHORT is the only allowed value. This parameter has meaning only when the CONFIRM option is used on TYPE and causes control to be returned only after the confirmation reply has been received. TPF does not support the LONG option defined by the LU 6.2 architecture.

## TPPCC PREPARE\_TO\_RECEIVE

### RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions”.

## Entry Requirements

- The conversation must be in send state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the PREPARE\_TO\_RECEIVE verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_PKPTRCV_INVTYPE             | ....         | 000000A1       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKPTRCV_BADSTATE            | ....         | 000000A3       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- If TYPE=CONFIRM is specified, TPF's EVENT and POST facility is used to suspend the ECB until the program receives the confirmation reply. Since the ECB is suspended, all unnecessary resources should be released before this verb is issued. Failure to do so can cause serious system degradation.

**Note:** You can avoid the problem of suspended ECBs by using the ACTIVATE\_ON\_CONFIRMATION verb instead of PREPARE\_TO\_RECEIVE. See "TPPCC ACTIVATE\_ON\_CONFIRMATION" on page 427 for more information.

- If the program does not receive a confirmation reply within a certain amount of time, the TPF/APPC support issues a DEALLOCATE TYPE=ABEND to terminate the conversation. The amount of time that the system waits is determined by the value you specify for the TPRECV parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- This verb can be issued only when the conversation is in send state. Upon successful completion of this verb, the conversation is in receive state.
- The remote transaction program enters the corresponding send state when it receives the SEND indication in the WHAT\_RECEIVED (WHATRCV) parameter (refer to "TPPCC RECEIVE" on page 462). The remote transaction program can then send data to the local TPF transaction program.
- See "Programming Considerations" on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

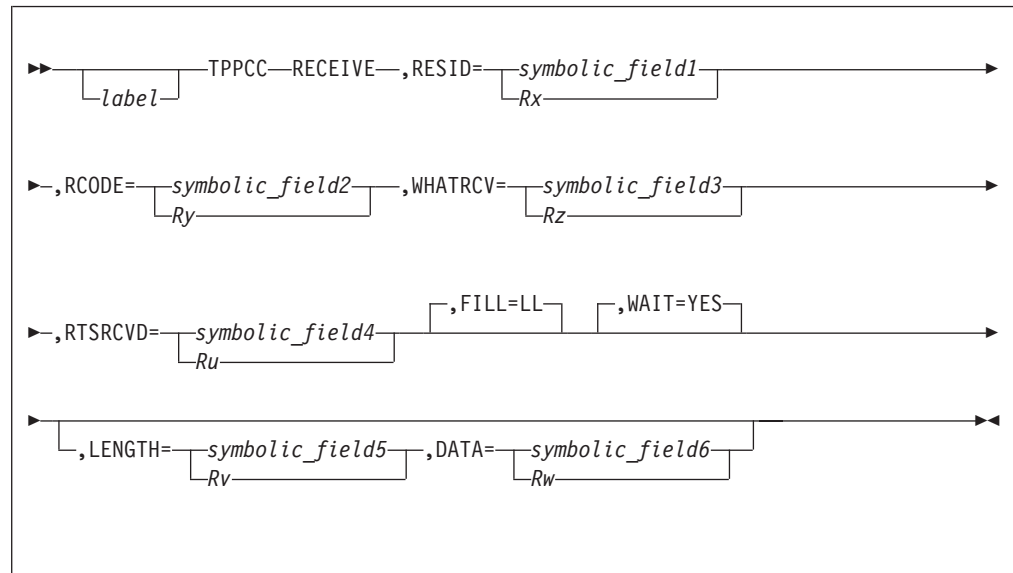
```
SYMB100 TPPCC PREPARE_TO_RECEIVE, X
 RESID=EBW004, X
 TYPE=SYNC, X
 LOCKS=SHORT, X
 RCODE=(R5)
```

## TPPCC RECEIVE

Use the TPPCC general macro with the RECEIVE verb specified to wait for information to arrive on the specified conversation and then receive the information. If information is already available, the information is received without waiting. The information received can be data, conversation status, or a confirmation request.

The transaction program can issue this verb when the conversation is in send state. In this case, the SEND indication is sent to the remote transaction program, and the local TPF side of the conversation is changed to receive state and waits for information to arrive. The remote program can send data to the local program after it receives the SEND indication.

## Format



*label*

A symbolic name can be assigned to the macro statement.

### RECEIVE

Directs the TPPCC macro to perform the RECEIVE verb function.

### RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

### FILL

LL specifies that the program is to receive one complete or truncated logical record, or a portion of a logical record that is equal to the length specified by the LENGTH parameter. See the programming considerations for an explanation of the record format.

TPF does not support the BUFFER option defined by the LU 6.2 architecture.

### WAIT

YES is the only supported option, which performs the function of

RECEIVE\_AND\_WAIT. TPF does not support the RECEIVE\_IMMEDIATE verb function defined by the LU 6.2 architecture.

## LENGTH<sup>2</sup>

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field in which the maximum length of data to receive is supplied. The value can be from 0 to 32767. When control is returned to the TP, this variable contains the actual amount of data the program received up to the maximum. If the TP receives information other than data, this variable remains unchanged.

## DATA<sup>2</sup>

Specifies the symbolic name of a field or a register pointing to a field in which the program is to receive the data.

## RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions" on page 464.

## WHATRCV

Specifies the symbolic name of a 1-byte field or a register that contains the WHAT\_RECEIVED indication. The WHAT\_RECEIVED indication is one of the following values:

### LU62WR\_DATACOMPLETE

Indicates that a complete logical record or the last remaining portion of a logical record is received.

### LU62WR\_DATAINCOMPLETE

Indicates that less than a complete logical record was received by the local TPF transaction program. This can be caused by the remote transaction program issuing a verb that causes data truncation, such as DEALLOCATE with one of the abend parameters.

### LU62WR\_LL\_TRUNCATED

Indicates that the 2-byte length field of a logical record was truncated after the first byte. The TPF/APPC support code discards the truncated length field; it is not received by the program.

### LU62WR\_CONFIRM

Indicates that the remote program issued CONFIRM, requesting the local TPF transaction program to respond by issuing CONFIRMED. The local TPF transaction program can instead issue SEND\_ERROR.

### LU62WR\_CONFIRMSEND

Indicates the remote program issued PREPARE\_TO\_RECEIVE with TYPE=CONFIRM. The local TPF transaction program can respond by issuing CONFIRMED or SEND\_ERROR.

### LU62WR\_SEND

Indicates the remote program has entered receive state, placing the local program in send state. The local TPF transaction program can now issue SEND\_DATA.

### LU62WR\_CONFIRMDLLOC

Indicates the remote program issued DEALLOCATE with TYPE=CONFIRM. The local TPF transaction program can respond by issuing CONFIRMED or SEND\_ERROR.

---

2. The LENGTH parameter is a supplied and returned parameter. If used, both DATA and LENGTH must be specified.

## TPPCC RECEIVE

### RTSRCVD

Specifies the symbolic name of a 1-byte field or a register that contains a value that indicates whether a REQUEST\_TO\_SEND has been received. The REQUEST\_TO\_SEND\_RECEIVED indication is:

#### LU62\_RTSND\_RCVYES

Indicates a REQUEST\_TO\_SEND indication has been received from the remote transaction program. The remote program issued REQUEST\_TO\_SEND, requesting the local TPF transaction program to enter receive state and placing the remote transaction program in send state.

Any other value indicates that a REQUEST\_TO\_SEND notification has not been received.

## Entry Requirements

- If you omit the DATA and LENGTH parameters, D0 must be available for use. RECEIVE passes data to the transaction program on this data level as described in the programming considerations.
- If you specify the DATA and LENGTH parameters, the entire area referenced by the DATA and LENGTH must be addressable by the TPF/APPC support code.
- The conversation must be in send or receive state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- If the LENGTH parameter was specified, the field can be updated to indicate the amount of data received.
- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the RECEIVE verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_INVALID_LENGTH              | ....         | 00000006       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKRECEV_BADSTATE            | ....         | 000000B1       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |



| Symbolic Name             | Primary Code | Secondary Code |
|---------------------------|--------------|----------------|
| LU62RC_CONV_TYPE_MISMATCH | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT  | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM    | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC    | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR    | 0008         |                |
| LU62RC_DLLOC_NORMAL       | 0009         |                |
| LU62RC_PGMERR_NOTRUNC     | 000C         |                |
| LU62RC_PGMERR_TRUNC       | 000D         |                |
| LU62RC_PGMERR_PURGING     | 000E         |                |
| LU62RC_CONVFAIL_RETRY     | 000F         |                |
| LU62RC_CONVFAIL_NORETRY   | 0010         |                |
| LU62RC_SVCERR_NOTRUNC     | 0011         |                |
| LU62RC_SVCERR_TRUNC       | 0012         |                |
| LU62RC_SVCERR_PURGING     | 0013         |                |
| LU62RC_TPF_ABEND          | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- The transaction program must check both the return code and the WHAT\_RECEIVED (WHATRCV) indicator.

**Note:** If the return code is not LU62RC\_OK, WHATRCV has no meaning and should not be examined.

- If you do not specify the LENGTH and DATA parameters and the WHAT\_RECEIVED indicator indicates that data is returned to the transaction program, then one of the following occurred:
  - The program received a complete logical record or the last remaining portion of a complete logical record.<sup>3</sup> The WHAT\_RECEIVED indicator is set to LU62WR\_DATACOMPLETE.
  - The program received an incomplete logical record because it was truncated.<sup>4</sup>

The data is presented in a 4KB working storage block on data level 0 in AMSG format (defined by the AM0SG DSECT).

If an entire logical record or the beginning of a truncated record is received, the complete logical record length is placed in the first 2 bytes of the logical record (AM0TXT), followed by the text of the record.

3. The last remaining portion of a complete logical record can be returned if this RECEIVE followed a RECEIVE that used the DATA and LENGTH parameters and had a WHAT\_RECEIVED indicator of LU62WR\_DATAINCOMPLETE.

4. Truncated logical records can be received if the remote transaction program caused data truncation by issuing SEND\_ERROR or DEALLOCATE TYPE=any ABEND.

## TPPCC RECEIVE

If the last remaining portion of a logical record is received, the text of the record is not preceded by a logical length field and begins at AM0TXT.

If the complete logical record does not fit in a single block, the remainder of the logical record is chained in 4KB, short-term file pool records. The standard TPF forward chain field (AM0FWD) contains the file address of the next segment of a logical record. A forward chain field containing zeros denotes the end of the chain. The standard TPF message length field (AM0CCT) contains the number of bytes of text in each of the segments.<sup>5</sup>

If the entire logical record (or the beginning of a truncated record) is received, the logical record length presented in the first 2 bytes of a logical record is the length of the complete logical record, while the contents of the AM0CCT field contains the physical length of each segment of a logical record. If the completion of a logical record is received, the sum of the segments' AM0CCT fields indicates the total amount of data received.

- If you specify the DATA and LENGTH parameters and the WHAT\_RECEIVED indicator indicates that data is returned to the transaction program, then one of the following occurred:
  - The program received a complete logical record or the last remaining portion of a complete logical record. The length of the record or portion of the record is equal to or less than the length specified on the LENGTH parameter. If the length received is less than the value pointed to by the LENGTH parameter, the value is updated to indicate the actual amount of data received. The WHAT\_RECEIVED indicator is set to LU62WR\_DATACOMPLETE.
  - The program received an incomplete logical record, which can occur because:
    - The length of the logical record is greater than the length specified by the LENGTH parameter; in this case the amount received equals the length specified.
    - Only a portion of the logical record is available because it was truncated, the portion being equal to or less than the length specified by the LENGTH parameter.<sup>4</sup>

The WHAT\_RECEIVED indicator is set to LU62WR\_DATAINCOMPLETE. The program must issue another RECEIVE (or possibly multiple RECEIVES) to receive the remainder of the logical record.

**Note:** A LENGTH parameter value of zero has no special significance. The type of information available is indicated by the RCODE parameter and the WHATRCV parameters, as usual. If data is available, the WHATRCV parameter contains LU62WR\_DATAINCOMPLETE, but the program receives no data.

- The maximum size of a logical record is 32,767.
- If there is no data available, TPF's EVENT and POST facility is used to suspend the ECB until data arrives. Since the ECB is suspended, all unnecessary resources should be released before this verb is issued. Failure to do so can cause serious system degradation.

**Note:** You can avoid the problem of suspended ECBs by using the ACTIVATE\_ON\_RECEIPT verb instead of RECEIVE. See "TPPCC ACTIVATE\_ON\_RECEIPT" on page 431 for more information.

---

5. The AM0CCT field is equal to the actual text count in this block plus 5, which is the length of the AM0SG filler. Thus the physical length of data in a block is (AM0CCT - 5).

## TPPCC RECEIVE

- If the program does not receive any data within a certain amount of time, the TPF/APPC support issues a DEALLOCATE TYPE=ABEND to terminate the conversation. The amount of time that the system waits is determined by the value you specify for the TPRECV parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

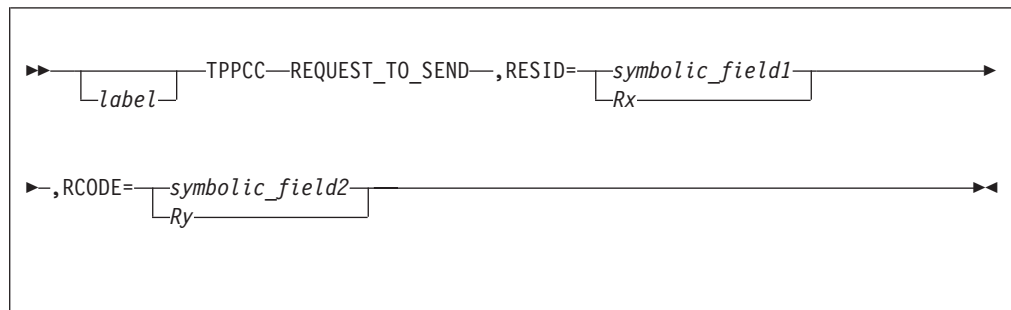
## Examples

```
SYMB100 TPPCC RECEIVE, X
 RESID=EBW004, X
 FILL=LL, X
 DATA=(R3), X
 LENGTH=(R2), X
 WAIT=YES, X
 RCODE=EBW064, X
 WHATRCV=EBW016, X
 RTSRCVD=EBSW01
```

## TPPCC REQUEST\_TO\_SEND

Use the TPPCC general macro with the REQUEST\_TO\_SEND verb specified to notify the remote transaction program that the local TPF transaction program is requesting to enter send state for the conversation. The conversation is changed to send state when the local TPF program receives a SEND indication from the remote program.

### Format



*label*

A symbolic name can be assigned to the macro statement.

### REQUEST\_TO\_SEND

Directs the TPPCC macro to perform the REQUEST\_TO\_SEND verb function.

### RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

### RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. the actual values returned are listed in “Return Conditions”.

### Entry Requirements

- The conversation must be in one of the following states:
  - send
  - receive
  - received-confirm
  - received-confirm-send
  - received-confirm-deallocate
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

### Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the REQUEST\_TO\_SEND verb. A

## TPPCC REQUEST\_TO\_SEND

complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_STATE_CHECK      | 0002         |                |
| LU62RC_SKRTSND_BADSTATE | ....         | 000000E1       |
| LU62RC_TPF_ABEND        | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- After issuing the REQUEST\_TO\_SEND verb, the local TPF transaction program must issue the RECEIVE verb to wait for the SEND indication to arrive from the remote program. The SEND indication is received in the WHATRCV parameter of the RECEIVE verb. (See “TPPCC RECEIVE” on page 462.)

For the local program to receive permission to send data, the remote program must do one of the following:

- Issue a RECEIVE verb
  - Issue a PREPARE\_TO\_RECEIVE verb
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

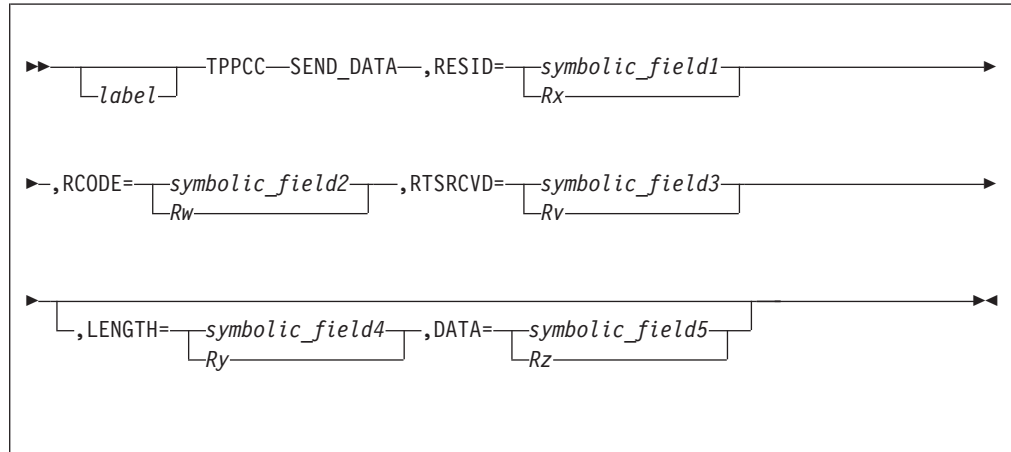
## Examples

```
SYMB100 TPPCC REQUEST_TO_SEND, X
 RESID=EBW004, X
 RCODE=EBW064
```

## TPPCC SEND\_DATA

Use the TPPCC general macro with the SEND\_DATA verb specified to send data to the remote transaction program. The data sent consists of logical records.

### Format



#### label

A symbolic name can be assigned to the macro statement.

#### SEND\_DATA

Directs the TPPCC macro to perform the SEND\_DATA verb function.

#### RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

#### DATA

Specifies the symbolic name of a field or a register pointing to a field from which the data is to be sent.

#### LENGTH

Specifies the symbolic name of a field or a register pointing to a field. This is a 2-byte field that contains the length of the data to be sent. The value can be from 0 to 32767. This data length is not related in any way to the length of a logical record. It is used only to determine the length of the data located at the address specified by the DATA parameter.

**Note:** If the DATA and LENGTH parameters are omitted, the data is assumed to be on data level 0 in AM0SG format. See the programming considerations for an explanation of the record format.

#### RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions" on page 471.

#### RTSRCVD

Specifies the symbolic name of a 1-byte field or a register that contains a value the indicates whether a REQUEST\_TO\_SEND has been received. The REQUEST\_TO\_SEND\_RECEIVED indication can contain:

**LU62\_RTSND\_RCVDYES**

Indicates a REQUEST\_TO\_SEND indication has been received from the remote transaction program. The remote program issued REQUEST\_TO\_SEND requesting the local TPF transaction program to enter receive state and placing the remote transaction program in send state.

Any other value indicates that a REQUEST\_TO\_SEND notification has not been received.

**Entry Requirements**

- The conversation must be in send state.
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

**Return Conditions**

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the SEND\_DATA verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_INVALID_LENGTH              | ....         | 00000006       |
| LU62RC_PKSENDD_BADLL               | ....         | 000000F1       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKSENDD_BADSTATE            | ....         | 000000F2       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |

## TPPCC SEND\_DATA

| Symbolic Name         | Primary Code | Secondary Code |
|-----------------------|--------------|----------------|
| LU62RC_SVCERR_PURGING | 0013         |                |
| LU62RC_TPF_ABEND      | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- The following considerations apply if you do not specify the DATA and LENGTH parameters:
  1. The data to be sent must be in the main storage block attached to data level 0 (D0) in AMMSG format. The block can be a small (381), large (1055), or 4KB storage block. The data must be in logical record format, that is, the first 2 bytes contain the length of the logical record followed by the data. The storage block can contain one or more full logical records, or it can contain a partial logical record. The number of bytes within the block to be sent is always in the AMMSG length field (AM0CCT) regardless of the logical record boundary. It is the transaction program's responsibility to insure the accuracy of both the logical record length field (LL preceding the text) and the storage block length field (AM0CCT).<sup>6</sup>
  2. If the complete logical record does not fit into one block, the message can be forward chained. The number of bytes of text for each of the chained segments is placed in the AMMSG length field (AM0CCT), and the file address of the next segment is placed in the standard TPF forward chain field (AM0FWD). A forward chain field containing zeros indicates the last segment of a chained logical record. The logical record length in the first 2 bytes of the first or only segment contains the length of the entire logical record, while the AM0CCT field of each segment contains the physical number of bytes of text in each segment.
  3. The transaction program can also send logical records by issuing multiple SEND\_DATA verbs for segments of the logical record. In this case, the first segment must contain the logical record length (LL) preceding the first byte of the text, and the AMMSG length field (AM0CCT) must contain the number of bytes within each storage block.
  4. If the return code is not LU62RC\_OK, the data remains on D0 unchanged.
- The following considerations apply if you specify the DATA and LENGTH parameters:
  1. The data must be in logical record format, that is, the first 2 bytes contain the length of the logical record followed by the data.
  2. The number of bytes to be sent is always the value pointed to by the LENGTH parameter regardless of the logical record boundary.
  3. It is the transaction program's responsibility to insure the accuracy of both the logical record length field (LL preceding the text) and the LENGTH parameter.
  4. The entire storage area referred to by the DATA and LENGTH parameters must be addressable by the TPF/APPC support code.
- The maximum size of a logical record is 32,767.

6. The AM0CCT field is equal to the actual text count in this block plus 5, which is the length of the AM0SG filler. Thus the physical length of data in a block is (AM0CCT - 5).



- When the value of the RTSRCVD parameter is LU62\_RTSND\_RCVYES, the remote program is requesting the local TPF transaction program to enter receive state and thereby place the remote program in send state. The local TPF transaction program enters receive state by issuing a RECEIVE verb or a PREPARE\_TO\_RECEIVE verb. The remote partner program enters the corresponding send state when it issues a RECEIVE verb and receives the SEND indicator on the WHATRCV parameter.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

```
SYMB100 TPPCC SEND_DATA, X
 RESID=(R6), X
 DATA=(R3), X
 LENGTH=(R2), X
 RCODE=EBW064, X
 RTSRCVD=EBSW01
```

## TPPCC SEND ERROR

Upon successful completion of this verb, the local TPF transaction program is in send state, and the remote program is in receive state. The transaction program's logic defines any further action.

Diagram illustrating the structure of the **SEND\_ERROR** command:

- The command is initiated by a **TPPCC—SEND\_ERROR** instruction.
- The **RESID** (Residual) field is defined by **symbolic\_field1** and register **Rx**.
- The **RCODE** (Return Code) field is defined by **symbolic\_field2** and register **Ry**.
- The **RTSRCVD** (Return Source Code) field is defined by **symbolic\_field3** and register **Rz**.
- Optional fields include **TYPE=PROG** and **LOGDATA=NO**.
- The command concludes with **TYPE=SVC**.

A symbolic name can be assigned to the macro statement.

Directs the TPPCC macro to perform the SEND ERROR verb function.

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

Specifies the type of error indication to be sent. The valid types are:

Specifies that the TPF transaction program detected an error that is to be reported to the remote transaction program.

Specifies that a TPF service program detected an error that is to be reported to the remote LU service program.

Specifies whether the error indication is logged. NO is the only supported option. TPF does not support the YES option defined by the LU 6.2 architecture.

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions” on page 475.

**RTSRCVD**

Specifies the symbolic name of a 1-byte field or a register that contains a value that indicates whether a REQUEST\_TO\_SEND has been received. The REQUEST\_TO\_SEND\_RECEIVED indication can contain:

**LU62\_RTSEND\_RCVDYES**

Indicates a REQUEST\_TO\_SEND indication has been received from the remote transaction program. The remote program issued REQUEST\_TO\_SEND requesting the local TPF transaction program to enter receive state and placing the remote transaction program in send state. Any other value indicates that a REQUEST\_TO\_SEND notification has not been received.

**Entry Requirements**

- The conversation must be in one of the following states:  
send  
receive  
received-confirm  
received-confirm-send  
received-confirm-deallocate
- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

**Return Conditions**

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following tables contain a list of the primary and secondary return codes that can be returned to the program that issued the SEND\_ERROR verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

Table 14. Return Codes for Conversation in SEND State

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_POSTED_DATA                 | ....         | 00000000       |
| LU62RC_POSTED_NODATA               | ....         | 00000001       |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_SKSENDE_BADSTATE            | ....         | 00000111       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |

## TPPCC SEND\_ERROR

Table 14. Return Codes for Conversation in SEND State (continued)

| Symbolic Name            | Primary Code | Secondary Code |
|--------------------------|--------------|----------------|
| LU62RC_SYNLVL_NOTSUPPORT | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM   | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC   | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR   | 0008         |                |
| LU62RC_PGMERR_PURGING    | 000E         |                |
| LU62RC_CONVFAIL_RETRY    | 000F         |                |
| LU62RC_CONVFAIL_NORETRY  | 0010         |                |
| LU62RC_SVCERR_PURGING    | 0013         |                |
| LU62RC_TPF_ABEND         | FFFF         |                |

Table 15. Return Codes for Conversation in RECEIVE State

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_DLLOC_NORMAL     | 0009         |                |
| LU62RC_CONVFAIL_RETRY   | 000F         |                |
| LU62RC_CONVFAIL_NORETRY | 0010         |                |
| LU62RC_TPF_ABEND        | FFFF         |                |

Table 16. Return Codes for Conversation in CONFIRM State.

| Symbolic Name           | Primary Code | Secondary Code |
|-------------------------|--------------|----------------|
| LU62RC_OK               | 0000         |                |
| LU62RC_PARAMETER_CHECK  | 0001         |                |
| LU62RC_PK_BAD_TCBID     | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID    | ....         | 00000002       |
| LU62RC_CONVFAIL_RETRY   | 000F         |                |
| LU62RC_CONVFAIL_NORETRY | 0010         |                |
| LU62RC_TPF_ABEND        | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb one that was assigned by an incoming ATTACH.
- The transaction programs can use this verb for various application level functions. For example, the program can issue this verb to truncate an incomplete logical record, to inform the remote program of an error it detected in data it received, or to reject a confirmation request.
- If the local TPF transaction program issues the SEND\_ERROR verb while in receive state, purging of all buffered input occurs. The incoming information that

is purged includes both data (logical records) and confirmation requests. In addition, if the confirmation request was due to a DEALLOCATE TYPE=CONFIRM verb issued by the remote transaction program, the DEALLOCATE request is also purged. After issuing the SEND\_ERROR verb from receive state, the local transaction program will be in send state, and the remote transaction program will be in receive state.

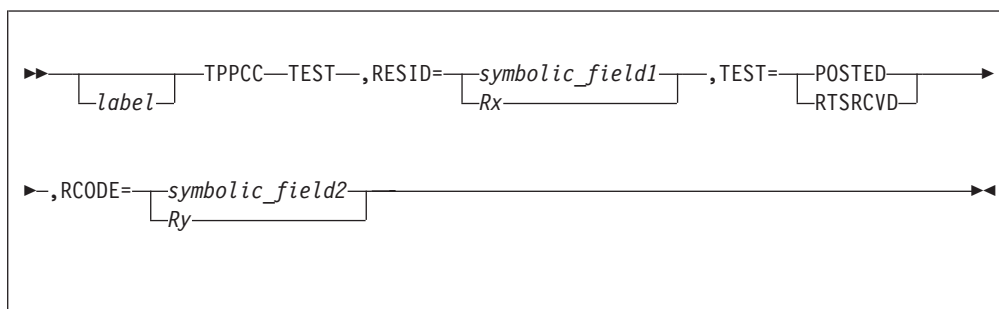
- When the value of the RTSRCVD parameter is LU62\_RTSND\_RCVYES, the remote program is requesting the local TPF transaction program to enter receive state and thereby place the remote program in send state. The local TPF transaction program enters receive state by issuing a RECEIVE verb or a PREPARE\_TO\_RECEIVE verb. The remote partner program enters the corresponding send state when it issues a RECEIVE verb and receives the SEND indicator on the WHATRCV parameter.
- This verb resets or cancels posting. If posting is active and the conversation has been posted, posting is reset. If posting is active and the conversation has not been posted, posting is canceled.
- The truncation implied by the use of this verb can cause any of the following to be truncated:
  - Data, sent by means of the SEND\_DATA verb
  - Confirmation requests, sent by means of CONFIRM, PREPARE\_TO\_RECEIVE, or DEALLOCATE
  - Deallocation request, sent by means of DEALLOCATE TYPE=CONFIRM.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

```
SYMB100 TPPCC SEND_ERROR, X
 RESID=(R3), X
 TYPE=PROG, X
 LOGDATA=NO, X
 RCODE=EBW064, X
 RTSRCVD=EBSW01
```

Use the TPPCC general macro with the TEST verb specified to test the conversation for a specified condition. The results of the test are indicated in the return code.

## Format



*label*

A symbolic name can be assigned to the macro statement.

## TEST

Directs the TPPCC macro to perform the TEST verb function.

## RESID

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID is supplied. This resource ID must be the one assigned on the initial ALLOCATE for this conversation or one that was assigned by an incoming ATTACH.

## TEST

Specifies the type of test to be performed. The valid types are:

**POSTED**

Specifies to test whether the conversation has been posted.

## RTSRCVD

Specifies to test whether the REQUEST\_TO\_SEND notification has been received from the remote transaction program.

## RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in “Return Conditions” on page 479.

## Entry Requirements

- The conversation must be in one of the following states, depending on the type of test requested:

| Type     | State required   |
|----------|------------------|
| POSTED   | <u>receive</u>   |
| RTSTRCVD | send or receive. |

- See “Entry Requirements” on page 421 for the entry requirements relating to the TPPCC macro in general.

## Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, “Return Codes”.
- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following tables contain a list of the primary and secondary return codes that can be returned to the program that issued the TEST verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

Table 17. Return Codes for TEST=POSTED

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_POSTED_DATA                 | ....         | 00000000       |
| LU62RC_POSTED_NOTDATA              | ....         | 00000001       |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_NOT_RCV_STATE               | ....         | 00000122       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_DLLOC_NORMAL                | 0009         |                |
| LU62RC_POSTING_NOTACTIV            | 000B         |                |
| LU62RC_PGMERR_NOTRUNC              | 000C         |                |
| LU62RC_PGMERR_TRUNC                | 000D         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |
| LU62RC_SVCERR_NOTRUNC              | 0011         |                |
| LU62RC_SVCERR_TRUNC                | 0012         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_LLRCV_UNSUCESSFUL           | 0014         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## TPPCC TEST

Table 18. Return Codes for TEST=RTSRCVD

| Symbolic Name            | Primary Code | Secondary Code |
|--------------------------|--------------|----------------|
| LU62RC_OK                | 0000         |                |
| LU62RC_PARAMETER_CHECK   | 0001         |                |
| LU62RC_PK_BAD_TCBID      | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID     | ....         | 00000002       |
| LU62RC_LLRCV_UNSUCESSFUL | 0014         |                |
| LU62RC_TPF_ABEND         | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The value supplied in RESID must be the resource ID returned by the ALLOCATE verb or one that was assigned by an incoming ATTACH.
- The TEST=POSTED option is valid only when the conversation is in receive state.
- The TEST=POSTED option is used in conjunction with the POST\_ON\_RECEIPT verb. The use of POST\_ON\_RECEIPT and TEST allows a program to continue its processing while waiting for information to become available. The TPF transaction program issues POST\_ON\_RECEIPT for one or more conversations and then issues TEST for each conversation to determine when information is available to be received.
- For the TEST=POSTED option, the return code indicates whether posting is active, whether the conversation has been posted, and whether the information available is data or not. The TPF transaction program must issue the RECEIVE verb to receive the information for a conversation that has been posted.
- The TEST=POSTED option returns LU62RC\_LLRCV\_UNSUCESSFUL if the conversation has not been posted.
- Posting is active for a conversation on which the POST\_ON\_RECEIPT verb has been issued and posting has not been reset or cancelled. See “TPPCC POST\_ON\_RECEIPT” on page 456 for more information on posting.
- The TEST=RTSRCVD option returns LU62RC\_OK if the REQUEST\_TO\_SEND has been received. Otherwise, the REQUEST\_TO\_SEND indication has not been received from the remote transaction program and LU62RC\_LLRCV\_UNSUCESSFUL is returned.
- To enter receive state, the local transaction program must issue the appropriate verb, such as RECEIVE or PREPARE\_TO\_RECEIVE, when the REQUEST\_TO\_SEND indication is received.
- See “Programming Considerations” on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

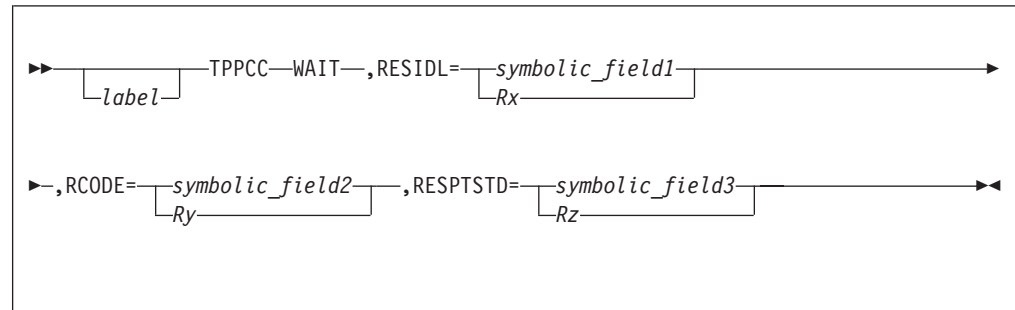
```
SYMB100 TPPCC TEST, X
 RESID=(R6), X
 TEST=RTSRCVD, X
 RCODE=EBW064
```



## TPPCC WAIT

Use the TPPCC general macro with the WAIT verb specified to wait for posting to occur on any mapped or basic conversation from a list of conversations. Posting of a conversation occurs when posting is active for the conversation and information that the conversation can receive, such as data, conversation status, or a request for confirmation is available.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### WAIT

Directs the TPPCC macro to perform the WAIT verb function.

#### RESIDL

Specifies the symbolic name of a field or a register pointing to a field. This is a variable length field that contains the resource IDs of the conversations for which posting is expected. The resource IDs must be the resource IDs assigned on the initial ALLOCATE for the conversations or assigned by an incoming ATTACH. The variable length field is in the format:

- Resource ID - 4 bytes repeated for each conversation
- Delimiter - 4 bytes of binary zeros.

**Note:** This list can contain as many as twenty entries, which allows space for nineteen resource IDs and the delimiter.

#### RCODE

Specifies the symbolic name of a field or a register pointing to a field. This is a 6-byte field in which the return code is placed. The return code consists of a 2-byte primary return code followed by a 4-byte secondary return code. The actual values returned are listed in "Return Conditions".

#### RESPSTD

Specifies the symbolic name of a field or a register pointing to a field. This is a 4-byte field in which the resource ID of a posted conversation is returned.

### Entry Requirements

- The conversations must be in receive state.
- See "Entry Requirements" on page 421 for the entry requirements relating to the TPPCC macro in general.

### Return Conditions

- The results of the verb are indicated by the value returned in RCODE. The valid return codes are listed in the following section, "Return Codes".

## TPPCC WAIT

- See “Return Conditions” on page 421 for the return conditions relating to the TPPCC macro in general.
- The following table contains a list of the primary and secondary return codes that can be returned to the program that issued the WAIT verb. A complete list of return codes and their definitions can be found in Table 11 on page 421 and in Table 12 on page 422.

| Symbolic Name                      | Primary Code | Secondary Code |
|------------------------------------|--------------|----------------|
| LU62RC_OK                          | 0000         |                |
| LU62RC_POSTED_DATA                 | ....         | 00000000       |
| LU62RC_POSTED_NOTDATA              | ....         | 00000001       |
| LU62RC_PARAMETER_CHECK             | 0001         |                |
| LU62RC_PK_BAD_TCBID                | ....         | 00000001       |
| LU62RC_PK_BAD_CONVID               | ....         | 00000002       |
| LU62RC_PK_EMPTY_RESIDL             | ....         | 00000003       |
| LU62RC_STATE_CHECK                 | 0002         |                |
| LU62RC_NOT_RCV_STATE               | ....         | 00000122       |
| LU62RC_ALLOC_ERROR                 | 0003         |                |
| LU62RC_TP_NOT_AVAIL_RETRY          | ....         | 084B6031       |
| LU62RC_TP_NOT_AVAIL_NO_RETRY       | ....         | 084C0000       |
| LU62RC_TPN_NOT_RECOGNIZED          | ....         | 10086021       |
| LU62RC_PIP_NOT_SPECIFIED_CORRECTLY | ....         | 10086032       |
| LU62RC_CONV_TYPE_MISMATCH          | ....         | 10086034       |
| LU62RC_SYNLVL_NOTSUPPORT           | ....         | 10086041       |
| LU62RC_DLLOC_ABEND_PGM             | 0006         |                |
| LU62RC_DLLOC_ABEND_SVC             | 0007         |                |
| LU62RC_DLLOC_ABEND_TMR             | 0008         |                |
| LU62RC_DLLOC_NORMAL                | 0009         |                |
| LU62RC_POSTING_NOTACTIV            | 000B         |                |
| LU62RC_PGMERR_NOTRUNC              | 000C         |                |
| LU62RC_PGMERR_TRUNC                | 000D         |                |
| LU62RC_PGMERR_PURGING              | 000E         |                |
| LU62RC_CONVFAIL_RETRY              | 000F         |                |
| LU62RC_CONVFAIL_NORETRY            | 0010         |                |
| LU62RC_SVCERR_NOTRUNC              | 0011         |                |
| LU62RC_SVCERR_TRUNC                | 0012         |                |
| LU62RC_SVCERR_PURGING              | 0013         |                |
| LU62RC_TPF_ABEND                   | FFFF         |                |

## Programming Considerations

- If you specify a register for a keyword value, the register must be enclosed in parentheses and in the range R1–R7.
- The values supplied in RESIDL must be the resource IDs returned by the ALLOCATE verb or IDs that were assigned by an incoming ATTACH.

- This verb is intended for use in conjunction with the POST\_ON\_RECEIPT verb. The use of these verbs allows a program to perform synchronous receiving from multiple conversations. The program issues POST\_ON\_RECEIPT for each conversation and then issues this verb for all the conversations in the resource list. The WAIT is satisfied when any conversation in the list has information available to be received.
- The value returned in RESPSTD is one of the resource IDs specified in RESIDL. The resource ID returned identifies the conversation that has been posted.
- Posting may or may not be active for the resource IDs specified in RESIDL. This verb waits for posting to occur on any conversation's resource ID in the list that has posting active.
- The return code indicates the posting state for the conversation identified in the RESPSTD parameter. A secondary return code of LU62RC\_POSTED\_DATA indicates that a data message is available for receipt. A secondary return code of LU62RC\_POSTED\_NOTDATA indicates that information other than a data message is available, such as a confirmation request of a SEND indication.
- Posting is reset for the conversation returned in the RESPSTD parameter. The TPF transaction program must issue the RECEIVE verb in order to receive the available information.
- This verb uses TPF's EVENT and POST facility to suspend the ECB until posting has occurred on one of the conversations. Since the ECB is suspended, all unnecessary resources should be released before this verb is issued. Failure to do so can cause serious system performance degradation.
- If posting does not occur within a certain amount of time, the TPF/APPC support issues a DEALLOCATE TYPE=ABEND to terminate all the conversations in the list. The amount of time that the system waits is determined by the value you specify for the TPWAIT parameter on the SNAKEY macro. See *TPF ACF/SNA Network Generation* for information about the SNAKEY macro.
- See "Programming Considerations" on page 426 for the programming considerations relating to the TPPCC macro in general.

## Examples

```
SYMB100 TPPCC WAIT, X
 RESIDL=EBW004, X
 RCODE=EBW064, X
 RESPSTD=(R7)
```

## TPRDC—Read a General Tape Record

This general macro causes the control program to obtain a block of storage and read the next record from a specified general tape into the block.

### Format

```
▶▶—label—TPRDC—NAME=yyy—,LEVEL=Dx—,BLOCK=Lx—▶▶
```

#### *label*

A symbolic name can 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 2 characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first 2 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 0–7, 14, or 15.

#### LEVEL=*Dx*

A symbolic data level (D0–DF) must be specified.

#### BLOCK=*Lx*

A symbolic storage block type (L0–L4) must be specified.

The following macro format is still supported:

```
▶▶—label—TPRDC—name—,—level—,BLOCK=Lx—
┌,BUF=NO
└,BUF=YES
```

#### *label*

A symbolic name can be assigned to the macro statement.

#### *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.

#### BLOCK=*Lx*

A symbolic storage block type (L0–L4) must be specified as the third parameter.

## Entry Requirements

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

- The general tape specified by this macro must be open when this macro is issued.
- A storage block must not be held by the ECB on the data level specified by this macro.

## 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 status of the Read operation is unknown.
- The file address reference word (FARW) on the data level specified by this macro is unchanged.
- The address of the storage block which contains the data record is placed in the core block reference word (CBRW) on the data level specified by this macro.
- When an End-of-Volume (EOV) condition occurs a tape switch will automatically be performed. For the TPRDC macro, AUTO=YES is implied for normal EOV conditions. Permanent error conditions and End-of-File (EOF) conditions will be reported through the WAITC error path and a tape switch will not be performed. (refer to the TDCTC macro in *TPF System Macros* or “TSYNC—Synchronize Tape” on page 490 for more information about the AUTO parameter).

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
***This practice is not recommended.***
- To ensure completion of the Read operation, a WAITC macro should be coded after this macro.
- The control program checks to ensure that the ECB is not holding a storage block on the data level specified by this macro, and that the general tape specified by this macro is open. If either condition is not met, control is transferred to the system error routine.
- The length of the record read from the general tape must be the same as the length of the storage block specified by this macro. If the lengths are not the same, an incorrect length error is raised and control transfers to the error label defined by the WAITC macro. The contents of ECB field CE1SUD indicate either a long length or a short length record error.

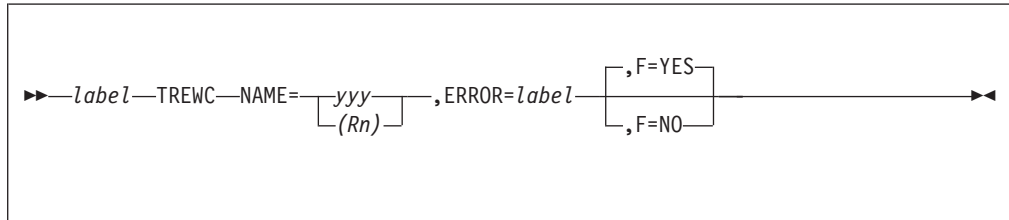
## Examples

None.

## TREWC–Rewind General Tape and Wait

This general macro positions the specified general tape at the beginning of its initial data record.

### Format



*label*

A symbolic name can 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 2 characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first 2 characters cannot be RT.

**(R*n*)**

The number of a register containing a pointer to the symbolic real-time tape name. *n* must be a decimal number from 0–7, 14, or 15.

**ERROR=*label***

The label of an operational program error routine within the current program segment must be specified.

**F** An optional keyword parameter can be specified to indicate whether fallback to the first volume of a multivolume file is to be allowed.

**YES**

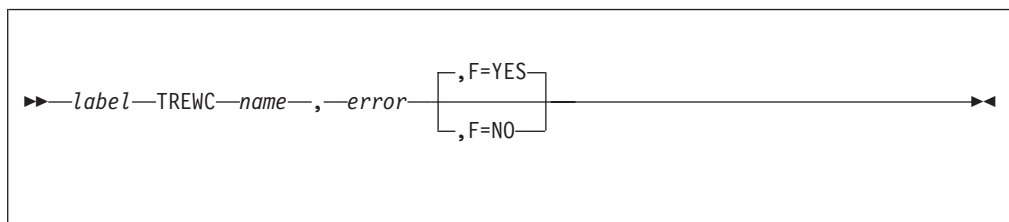
Fallback is to be allowed.

**NO**

Fallback is to be inhibited.

If omitted, a default of YES is assumed.

The following macro format is still supported:



*label*

A symbolic name can be assigned to the macro statement.

*name*

A 3-character symbolic general tape name must be specified as the first parameter.

*error*

The label of an operational program error routine within the current program segment must be specified as the second parameter.

- F** An optional keyword parameter can be specified to indicate whether fallback to the first volume of a multivolume file is to be allowed.

**YES**

Fallback is to be allowed.

**NO**

Fallback is to be inhibited.

If omitted, a default of YES is assumed.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must be open when this macro is issued.

## Return Conditions

- If no I/O hardware errors or unusual conditions have occurred, control is returned to the next sequential instruction (NSI). Otherwise control is transferred to the address specified by this macro.
- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.
- All pending I/O operations, including this request, are complete for this ECB.
- For I/O hardware errors, the system error routine has taken a storage dump and informed CRAS.

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
**This practice is not recommended.**
- If any I/O hardware errors have occurred, detailed information has been stored in the ECB.

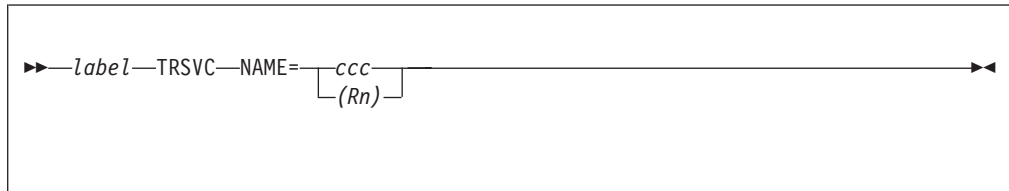
## Examples

None.

## TRSVC—Reserve General Tape

This general macro reserves the specified general tape for use by some future ECB. The tape will retain its current positioning until the future ECB takes control with a TASNC macro.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic general tape name. It can be:

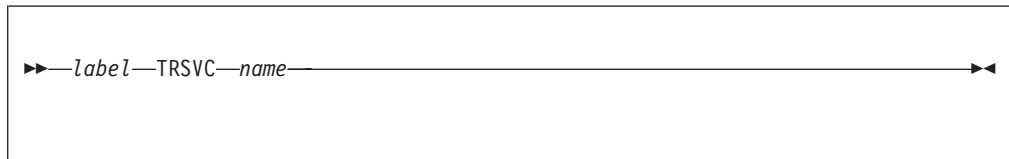
*ccc*

A 3-character string representing a symbolic general tape name. The first 2 characters must be alphabetic, and the third character must be alphabetic or numeric. For general tapes, the first 2 characters cannot be RT.

**(R*n*)**

The number of a register containing a pointer to the symbolic general tape name. *n* must be a decimal number from 0–7, 14, or 15.

The following macro format is still supported:



*label*

A symbolic name can be assigned to the macro statement.

*name*

A 3-character symbolic general tape name must be specified as the first parameter.

### Entry Requirements

- R9 must contain the address of the ECB being processed.
- The general tape specified by this macro must be open when this macro is issued.

### 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 executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
***This practice is not recommended.***
- Only a TASNC macro may be issued to a general tape after a TRSVC macro is issued.
- A TCLSC macro or TRSVC macro must be issued for all open tapes prior to issuing an EXITC macro.

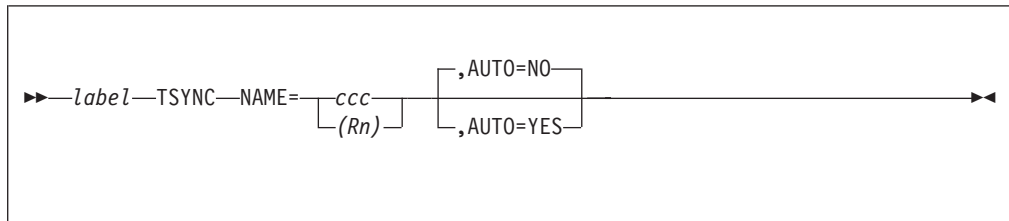
## Examples

None.

## TSYNC—Synchronize Tape

This general macro synchronizes a real-time or general tape. Tape synchronization ensures that the position of the tape, as detected by the device, is the same as detected by the application. This eliminates any discrepancies introduced by buffering; either logical (that is, tapes mounted in blocked mode) or physical (that is, tapes mounted on a device operating in buffered mode).

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### NAME

Specifies the symbolic general or real-time tape name. It can be:

*ccc*

A 3-character string representing a symbolic general or real time tape name. The first 2 characters must be alphabetic, and the third character must be alphabetic or numeric.

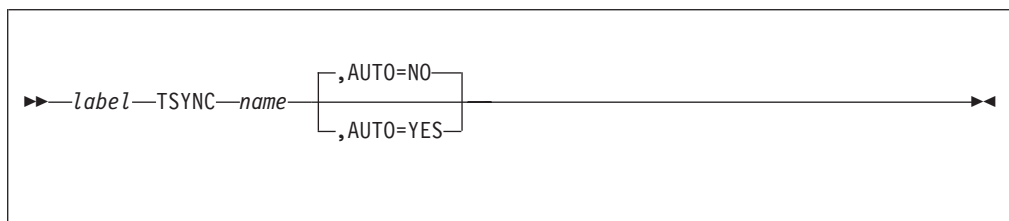
*(Rn)*

The number of a register containing a pointer to the symbolic real-time tape name. *n* must be a decimal number from 0 to 7, 14 or 15.

#### AUTO=NO|YES

An optional keyword parameter may be specified indicating whether a tape switch is to occur on a synchronize error. If specified, this parameter must be coded as YES if tape switch is to occur or as NO if tape switch is not to occur. If the tape is blocked, this parameter is forced to YES, even if it was coded as NO or omitted. If the tape is not blocked and the parameter is omitted, a default of NO is assumed.

The following macro format is still supported:



*label*

A symbolic name can be assigned to the macro statement.

*name*

A 3-character symbolic tape name must be specified as the first parameter.

#### AUTO=NO|YES

An optional keyword parameter may be specified indicating whether a tape

switch is to occur on a synchronize error. If specified, this parameter must be coded as YES if tape switch is to occur or as NO if tape switch is not to occur. If the tape is blocked, this parameter is forced to YES, even if it was coded as NO or omitted. If the tape is not blocked and the parameter is omitted, a default of NO is assumed.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- If the symbolic tape name specified by this macro is that of a general tape, that tape must be open when this macro is issued.

## 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 status of the synchronize operation is unknown.

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
***This practice is not recommended.***
- To ensure completion of the synchronization operation, a WAITC macro should be coded after this macro.
- If the tape is a blocked tape, the logical buffer is first synchronized by issuing a write command to write out any data contained in the logical buffer.

**Note:** If the device is not capable of operating in buffered mode, the synchronize command code (X'43') is converted to a NOP command code (X'03').

- Synchronization is simulated for a nonbuffered device by issuing an NOP command.
- Upon successful completion of the synchronization operation both the logical buffer and the physical buffer (if any) are empty.
- If the synchronization fails and a tape switch occurs, the records are recovered from the buffer and written to the new active tape. If the synchronization fails and a tape switch does not occur, the records are placed on top of the module queue.
- If there is a possibility of the ECB having outstanding DASD I/O, a WAITC macro should be coded before the TSYNC macro to ensure that the TSYNC macro will signal tape errors only.
- If this macro is issued prior to the end of tape restart, the ECB is exited and a system error issued.

## Examples

None.

## TWRTC—Write a General Tape Record

This general macro writes a record from a storage block to the specified general tape. The block of storage containing the record is detached from the ECB and returned to the appropriate pool.

### Format

```

▶▶—label—TWRTC—NAME=—yyy—, LEVEL=Dx—▶▶
 └─(Rn)─┘

```

#### *label*

A symbolic name can 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 2 characters must be alphabetic and the third character must be alphabetic or numeric. For general tapes, the first 2 characters cannot be RT.

#### **(R*n*)**

The number of a register containing a pointer to the symbolic real-time tape name. *n* must be a decimal number from 0–7, 14 or 15.

#### **LEVEL=*Dx***

A symbolic data level (D0–DF) must be specified.

The following macro format is still supported:

```

▶▶—label—TWRTC—name—, —Dx—▶▶

```

#### *label*

A symbolic name can be assigned to the macro statement.

#### *name*

A symbolic general tape name must be specified as the first parameter.

*Dx* 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.
- A storage block must be held by the ECB on the data level specified by **LEVEL**.
- The general tape specified by this macro must be open when this macro is issued.

## 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 file address reference word on the data level specified by **LEVEL** is unchanged.
- The core block reference word on the data level specified by **LEVEL** is updated to indicate that the storage block is no longer held by the ECB.
- When a condition such as End-of-Volume (EOV) or permanent error occurs, a tape switch will automatically occur. For the TWRTC macro, AUTO=YES is implied. (See TDCTC macro in *TPF System Macros* or “TSYNC—Synchronize Tape” on page 490 for more information about the AUTO parameter).

## Programming Considerations

- This macro can be executed on any I-stream.
- Both keyword and positional parameters may be used in the same macro call.  
***This practice is not recommended.***
- The control program checks to ensure that the ECB is holding a storage block on the data level specified by this macro and that the general tape name specified by this macro is open. If either condition is not met, control is transferred to the system error routine.
- The status of the Write operation can never be determined by the operational program.
- The operational program may use the request level specified by this macro immediately upon return from the control program.
- The contents of the entire storage block are written to tape.
- When writing to an unblocked general tape mounted on a buffered device, the mode of operation (buffered or Tape Write Immediate) is determined by the setting of the tertiary status byte in the tape status table.

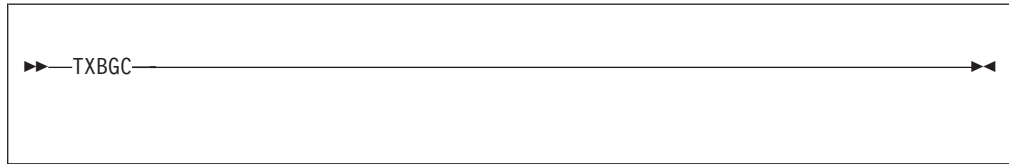
## Examples

None.

## TXBGC—Begin a Global Transaction

This general macro begins a global transaction.

### Format



### Entry Requirements

This general macro is restricted to ECB-controlled programs. Register 9 (R9) has to be the base of a valid entry control block (ECB) and must be executing in the ECB virtual memory (EVM).

### Return Conditions

- Control is returned to the next sequential instruction.
- R15 contains the return code.
- If the return code is 0, the macro has completed successfully.
- If the return code is -1, resource managers are not opened.
- 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.
- Resource managers that are compliant with the X/Open interface must be successfully opened to be included in the global transaction.

### Examples

The following example begins a transaction that reads, updates, and writes two records, and then commits the transaction.

```

 TXBGC , Begin transaction
*
 FIWHC D1,ERROR1 Read and lock record 1
 FIWHC D2,ERROR1 Read and lock record 2
*
 L R3,CE1CR1
 L R4,CE1CR2
 MVC USER2(,R3),USER1(R4) Updates to record 1 go here
 MVC USER2(,R4),USER1(R3) Updates to record 2 go here
*
 MVC CE1FA1(2),0(R3)
 FILUC D1 Write and unlock record 1
*
 MVC CE1FA2(2),0(R4)
 FILUC D2 Write and unlock record 2
*
 TXCMC , Commit the transaction

```

The following example begins a transaction that reads, updates, and writes two records, and then begins another nested transaction that reads, updates, and writes

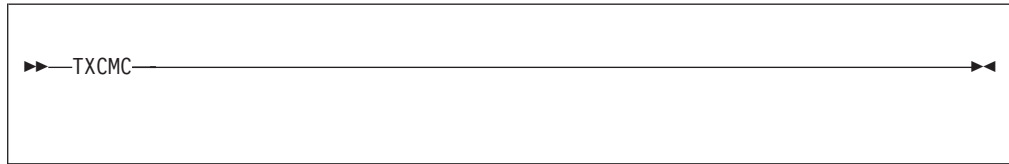
additional records. The nested transaction is committed. The root transaction performs additional processing and then commits the transaction.

|   |                          |                               |
|---|--------------------------|-------------------------------|
|   | TXBGC ,                  | Begin transaction             |
| * |                          |                               |
|   | FIWHC D1,ERROR1          | Read and lock record 1        |
|   | FIWHC D2,ERROR1          | Read and lock record 2        |
| * |                          |                               |
|   | L R3,CE1CR1              |                               |
|   | L R4,CE1CR2              |                               |
|   | MVC USER2(,R3),USER1(R4) | Updates to record 1 go here   |
|   | MVC USER2(,R4),USER1(R3) | Updates to record 2 go here   |
| * |                          |                               |
|   | MVC CE1FA1(2),0(R3)      |                               |
|   | FILUC D1                 | Write and unlock record 1     |
| * |                          |                               |
|   | MVC CE1FA2(2),0(R4)      |                               |
|   | FILUC D2                 | Write and unlock record 2     |
| * |                          |                               |
|   | TXBGC ,                  | Begin a nested transaction    |
| * |                          |                               |
|   | FIWHC D3,ERROR2          | Read and lock record 3        |
|   | FIWHC D4,ERROR2          | Read and lock record 4        |
| * |                          |                               |
|   | L R3,CE1CR3              |                               |
|   | L R4,CE1CR4              |                               |
|   | MVC USER2(,R3),USER1(R4) | Updates to record 3 go here   |
|   | MVC USER2(,R4),USER1(R3) | Updates to record 4 go here   |
| * |                          |                               |
|   | MVC CE1FA3(2),0(R3)      |                               |
|   | FILUC D3                 | Write and unlock record 3     |
| * |                          |                               |
|   | MVC CE1FA4(2),0(R4)      |                               |
|   | FILUC D4                 | Write and unlock record 4     |
| * |                          |                               |
|   | TXCMC ,                  | Commit the nested transaction |
| * |                          |                               |
|   | FIWHC D5,ERROR1          | Read and lock record 5        |
| * |                          |                               |
|   | L R3,CE1CR5              |                               |
|   | MVC USER3(,R3),=C'DONE'  | Updates to record 5 go here   |
| * |                          |                               |
|   | MVC CE1FA5(2),0(R3)      |                               |
|   | FILUC D5                 | Write and unlock record 5     |
| * |                          |                               |
|   | TXCMC ,                  | Commit the root transaction   |

## TXCMC—Commit a Global Transaction

This general macro commits a global transaction.

### Format



### Entry Requirements

This macro is restricted to ECB-controlled programs. Register 9 (R9) has to be the base of a valid entry control block (ECB) and must be executing in the ECB virtual memory (EVM).

### Return Conditions

- Control is returned to the next sequential instruction.
- R15 contains the return code.
- If the return code is 0, the macro has completed successfully.
- If the return code is -5 (that is, TX\_PROTOCOL\_ERROR), the TXBGC macro or the tx\_begin C function has not been issued by this ECB.
- 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.
- If the TXCMC macro is issued from a nested transaction, the work of the transaction will be rolled into the next higher level transaction.
- If the TXCMC macro is issued from a root transaction, the work of the transaction will be committed.

### Examples

The following example begins a transaction that reads, updates, and writes two records, and then commits the transaction.

```

 TXBGC , Begin transaction
*
 FIWHC D1,ERROR1 Read and lock record 1
 FIWHC D2,ERROR1 Read and lock record 2
*
 L R3,CE1CR1
 L R4,CE1CR2
 MVC USER2(,R3),USER1(R4) Updates to record 1 go here
 MVC USER2(,R4),USER1(R3) Updates to record 2 go here
*
 MVC CE1FA1(2),0(R3)
 FILUC D1 Write and unlock record 1
*
 MVC CE1FA2(2),0(R4)
 FILUC D2 Write and unlock record 2
*
 TXCMC , Commit the transaction

```



The following example begins a transaction that reads, updates, and writes two records, and then begins another nested transaction that reads, updates, and writes additional records. The nested transaction is committed. The root transaction performs additional processing and then commits the transaction.

```

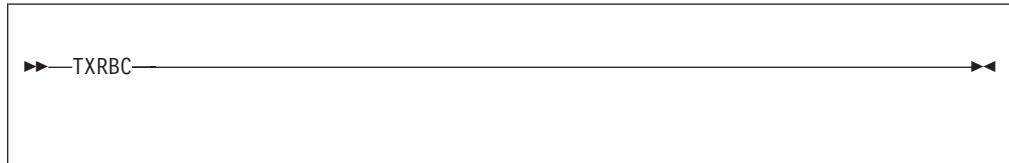
 TXBGC , Begin transaction
*
 FIWHC D1,ERROR1 Read and lock record 1
 FIWHC D2,ERROR1 Read and lock record 2
*
 L R3,CE1CR1
 L R4,CE1CR2
 MVC USER2(,R3),USER1(R4) Updates to record 1 go here
 MVC USER2(,R4),USER1(R3) Updates to record 2 go here
*
 MVC CE1FA1(2),0(R3)
 FILUC D1 Write and unlock record 1
*
 MVC CE1FA2(2),0(R4)
 FILUC D2 Write and unlock record 2
*
 TXBGC , Begin a nested transaction
*
 FIWHC D3,ERROR2 Read and lock record 3
 FIWHC D4,ERROR2 Read and lock record 4
*
 L R3,CE1CR3
 L R4,CE1CR4
 MVC USER2(,R3),USER1(R4) Updates to record 3 go here
 MVC USER2(,R4),USER1(R3) Updates to record 4 go here
*
 MVC CE1FA3(2),0(R3)
 FILUC D3 Write and unlock record 3
*
 MVC CE1FA4(2),0(R4)
 FILUC D4 Write and unlock record 4
*
 TXCMC , Commit the nested transaction
*
 FIWHC D5,ERROR1 Read and lock record 5
*
 L R3,CE1CR5
 MVC USER3(,R3),=C'DONE' Updates to record 5 go here
*
 MVC CE1FA5(2),0(R3)
 FILUC D5 Write and unlock record 5
*
 TXCMC , Commit the root transaction

```

## TXRBC–Roll Back a Global Transaction

This general macro rolls back the work of the transaction activated by the entry control block (ECB) of the caller.

### Format



### Entry Requirements

This macro is restricted to ECB-controlled programs. Register 9 (R9) has to be the base of a valid ECB and it must be executing in the ECB virtual memory (EVM).

### Return Conditions

- Control is returned to the next sequential instruction.
- R15 contains the return code.
- If the return code is 0, the macro has completed successfully.
- If the return code is -5 (that is, TX\_PROTOCOL\_ERROR), the TXBGC macro or the tx\_begin C function has not been issued by this ECB.
- 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.
- Return code -5 (that is, TX\_PROTOCOL\_ERROR) will be returned if the TXBGC macro has not been issued by this ECB.

### Examples

The following example begins a transaction that reads, updates, and writes two records, and then rolls back the transaction.

```

 TXBGC , Begin transaction
*
 FIWHC D1,ERROR1 Read and lock record 1
 FIWHC D2,ERROR1 Read and lock record 2
*
 L R3,CE1CR1
 L R4,CE1CR2
 MVC USER2(,R3),USER1(R4) Updates to record 1 go here
 MVC USER2(,R4),USER1(R3) Updates to record 2 go here
*
 MVC CE1FA1(2),0(R3)
 FILUC D1 Write and unlock record 1
*
 MVC CE1FA2(2),0(R4)
 FILUC D2 Write and unlock record 2
*
* application logic to perform other work as part of this
* transaction goes here
*
* application logic that detects an error and decides that

```

```

* a rollback should occur goes here
*
 TXRBC , Roll back the transaction

```

The following example begins a transaction that reads, updates, and writes two records, and then begins a nested transaction that reads, updates, and writes additional records. The nested transaction detects an error and rolls back the transaction. The root transaction also rolls back the transaction.

```

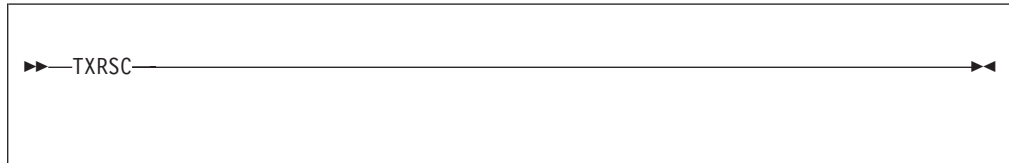
 TXBGC , Begin transaction
*
 FIWHC D1,ERROR1 Read and lock record 1
 FIWHC D2,ERROR1 Read and lock record 2
*
 L R3,CE1CR1
 L R4,CE1CR2
 MVC USER2(,R3),USER1(R4) Updates to record 1 go here
 MVC USER2(,R4),USER1(R3) Updates to record 2 go here
*
 MVC CE1FA1(2),0(R3)
 FILUC D1 Write and unlock record 1
*
 MVC CE1FA2(2),0(R4)
 FILUC D2 Write and unlock record 2
*
 TXBGC , Begin a nested transaction
*
 FIWHC D3,ERROR2 Read and lock record 3
 FIWHC D4,ERROR2 Read and lock record 4
*
 L R3,CE1CR3
 L R4,CE1CR4
 MVC USER2(,R3),USER1(R4) Updates to record 3 go here
 MVC USER2(,R4),USER1(R3) Updates to record 4 go here
*
 MVC CE1FA3(2),0(R3)
 FILUC D3 Write and unlock record 3
*
 MVC CE1FA4(2),0(R4)
 FILUC D4 Write and unlock record 4
*
 application logic to perform other work as part of this
 transaction goes here
*
 application logic that detects an error and decides that
 a rollback of the nested transaction should occur goes here
*
 TXRBC , Roll back the nested transaction
*
 application logic that detects an error and decides that
 a rollback of the root transaction should occur goes here
*
 TXRBC , Roll back the root transaction

```

## TXRSC—Resume a Global Transaction

This general macro continues a transaction that was suspended by the entry control block (ECB) of the caller.

### Format



### Entry Requirements

This macro is restricted to ECB-controlled programs. Register 9 (R9) has to be the base of a valid ECB and must be executing in the ECB virtual memory (EVM).

### Return Conditions

- Control is returned to the next sequential instruction.
- R15 contains the return code.
- If the return code is 0, the macro has completed successfully.
- If the return code is -5 (that is, TX\_PROTOCOL\_ERROR), the TXBGC macro or tx\_begin C function has not been issued by this ECB.
- If the return code is -7 (that is, TX\_FATAL), the TXSPC macro or tx\_suspend\_tpf C function has not been issued by this ECB.
- 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 TPF system permits the TXSPC macro to be issued while the transaction is already suspended. Each suspend request increments a suspend counter associated with the transaction by 1.
- The TXRSC macro decrements the suspend counter associated with the transaction by 1. The transaction resumes only when the counter reaches zero.

### Examples

The following example shows some of the ways that the TXSPC macro (suspend a global transaction) and the TXRSC macro (resume a global transaction) can be used in either root transaction or nested transaction scenarios. The comments to the right of the example code describe which transactions are active and what value is contained in the suspend counter throughout the example.

Transaction 1 is a root transaction that is suspended using the TXSPC macro before beginning transaction 2 (another root transaction). Transaction 2 is then suspended twice and resumed twice. After the transaction has continued (TXRSC) for the second time, the suspend counter is zero so transaction 2 becomes active again. Transaction 2 continues processing and is eventually committed (TXCMC). Transaction 1 remains suspended until it is explicitly resumed by the next TXRSC macro.

Transaction 3 then begins as a nested transaction under transaction 1. The transaction manager (TM) calls the TXSPC macro to suspend both transaction 3 and transaction 1.

**Note:** At this point, no transaction is active and any records filed here are outside of commit scope protection.

Later, the TM calls the TXRSC macro to continue the transactions. Eventually, the TM calls the TXCMC macro to commit first the nested and then the root transactions.

```

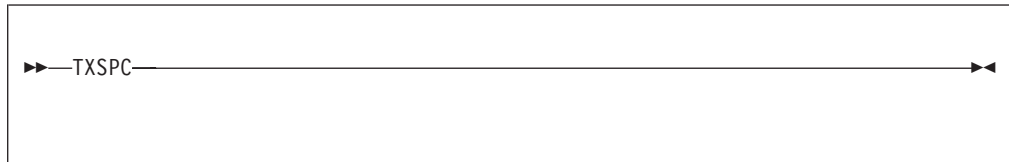
TXBGC /* begin transaction 1, root */
.
TXSPC /* suspend transaction 1, */
. /* suspend counter = 1. */
.
TXBGC /* begin transaction 2, root */
. /* not a nested transaction */
. /* transaction 2 is now active */
.
TXSPC /* suspend transaction 2 */
. /* suspend counter = 1. */
.
TXSPC /* suspend transaction 2 again */
. /* suspend counter = 2. */
.
TXRSC /* resume transaction 2 */
. /* suspend counter = 1. */
. /* transaction is still suspended*/
.
TXRSC /* resume transaction 2 */
. /* suspend counter = 0. */
. /* transaction 2 is now active */
.
TXCMC /* commit transaction 2 */
.
. /* transaction 1 is suspended */
. /* suspend counter is 1. */
.
TXRSC /* transaction 1 is now active */
. /* suspend counter is 0. */
.
TXBGC /* begin transaction 3, nested */
.
.
TXSPC /* suspend transaction 3 and 1 */
. /* suspend counter is 1. */
.
TXRSC /* resume transaction 3 and 1 */
. /* suspend counter is 0. */
. /* transaction 3 is now active */
.
TXCMC /* commit transaction 3 */
. /* transaction 1 is now active */
.
TXCMC /* commit transaction 1 */

```

## TXSPC—Suspend a Global Transaction

This general macro is used to suspend the transaction that the calling entry control block (ECB) is in. Any work that is done by the caller's ECB is not considered part of the suspended transaction until a TXRSC macro is called.

### Format



### Entry Requirements

This macro is restricted to ECB-controlled programs. Register 9 (R9) has to be the base of a valid ECB and must be executing in the ECB virtual memory (EVM).

### Return Conditions

- Control is returned to the next sequential instruction.
- R15 contains the return code.
- If the return code is 0, the macro has completed successfully.
- If the return code is -5 (that is, TX\_PROTOCOL\_ERROR), the TXBGC macro or tx\_begin C function has not been issued by this ECB.
- 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.
- Return code -5 (that is, TX\_PROTOCOL\_ERROR) is returned if the TXBGC macro has not been issued by this ECB.
- The TPF system permits the TXSPC macro to be issued while the transaction is already suspended. Each suspend request increments a suspend counter associated with the transaction by 1.
- The TXRSC macro decrements the suspend counter associated with the transaction by 1. The transaction resumes only when the counter reaches zero.
- An independent transaction is created if a TXBGC macro is called while the current transaction is suspended.

### Examples

The following example shows some of the ways that the TXSPC macro (suspend a global transaction) and the TXRSC macro (resume a global transaction) can be used in either root transaction or nested transaction scenarios. The comments to the right of the example code describe which transactions are active and what value is contained in the suspend counter throughout the example.

Transaction 1 is a root transaction that is suspended using the TXSPC macro before beginning transaction 2 (another root transaction). Transaction 2 is then suspended twice and resumed twice. After the transaction has continued (TXRSC) for the second time, the suspend counter is zero so transaction 2 becomes active

again. Transaction 2 continues processing and is eventually committed (TXCMC). Transaction 1 remains suspended until it is explicitly resumed by the next TXRSC macro.

Transaction 3 then begins as a nested transaction under transaction 1. The transaction manager (TM) calls the TXSPC macro to suspend both transaction 3 and transaction 1.

**Note:** At this point, no transaction is active and any records filed here are outside of commit scope protection.

Later, the TM calls the TXRSC macro to continue the transactions. Eventually, the TM calls the TXCMC macro to commit first the nested and then the root transactions.

```

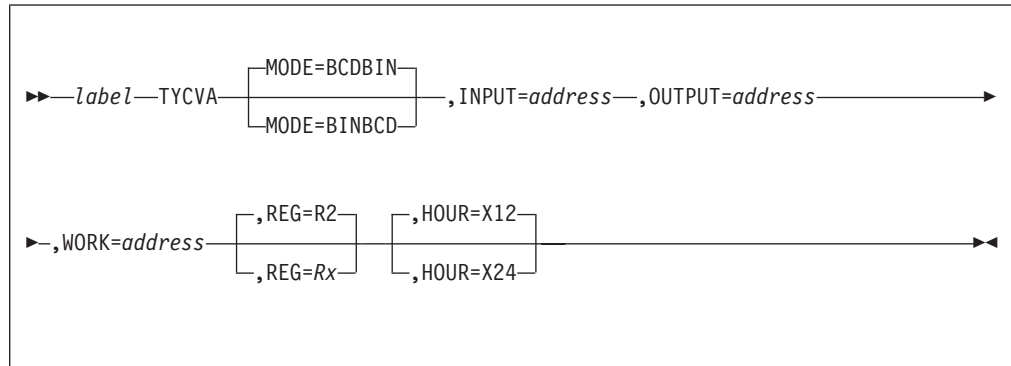
TXBGC /* begin transaction 1, root */
.
TXSPC /* suspend transaction 1, */
. /* suspend counter = 1. */
.
TXBGC /* begin transaction 2, root */
. /* not a nested transaction */
. /* transaction 2 is now active */
.
TXSPC /* suspend transaction 2 */
. /* suspend counter = 1. */
.
TXSPC /* suspend transaction 2 again */
. /* suspend counter = 2. */
.
TXRSC /* resume transaction 2 */
. /* suspend counter = 1. */
. /* transaction is still suspended*/
.
TXRSC /* resume transaction 2 */
. /* suspend counter = 0. */
. /* transaction 2 is now active */
.
TXCMC /* commit transaction 2 */
.
. /* transaction 1 is suspended */
. /* suspend counter is 1. */
.
TXRSC /* transaction 1 is now active */
. /* suspend counter is 0. */
.
.
TXBGC /* begin transaction 3, nested */
.
.
TXSPC /* suspend transaction 3 and 1 */
. /* suspend counter is 1. */
.
TXRSC /* resume transaction 3 and 1 */
. /* suspend counter is 0. */
. /* transaction 3 is now active */
.
TXCMC /* commit transaction 3 */
. /* transaction 1 is now active */
.
TXCMC /* commit transaction 1 */

```

## TYCVA—Time Conversion

This general macro converts the time of day from a 2-byte binary representation to a 5-byte EBCDIC representation, or vice versa. The EBCDIC time may be based on either the 12- or 24-hour clock.

### Format



*label*

A symbolic name can be assigned to the macro statement.

#### MODE=BCDBIN

EBCDIC time is converted to binary. This is the default if MODE is omitted.

##### INPUT=address

For MODE=BCDBIN, this is the symbolic address of a 5-byte field (no alignment necessary) which contains the following.

- Bytes 0–1 - EBCDIC hour
- Bytes 2–3 - EBCDIC minutes
- Byte 4 (12-hour clock only) - an alpha character indicating AM(A), NOON(N), PM(P), or MIDNIGHT(M). 1200A is treated as midnight and 1200P is treated as noon.

##### OUTPUT=address

For MODE=BCDBIN, this is the symbolic address of a 2-byte field (no alignment necessary) which will contain the following upon return from this macro.

- Byte 0 - binary representation of hour of 24-hour clock. (Midnight is returned as X'00'.)
- Byte 1 - binary representation of minutes.

#### MODE=BINBCD

Binary time is converted to EBCDIC.

##### INPUT=address

For MODE=BINBCD, this is the symbolic address of a 2-byte field (no alignment necessary) which contains the following.

- Byte 0 - binary representation of hour of 24-hour clock.
- Byte 1 - binary representation of minutes.

##### OUTPUT=address

For MODE=BINBCD, this is the symbolic address of a 5-byte field (no alignment necessary) which will contain the following upon return from this macro.



- Bytes 0-1 - EBCDIC hour in either 12- or 24-hour clock format as requested.
- Bytes 2-3 - EBCDIC minutes.
- Byte 4 (12-hour clock only) - an alpha character indicating AM(A), NOON(N), or PM(P). NOON(N) is considered 1200 only. For example, one minute after noon will be returned as 1201P.

**HOURL**

The EBCDIC clock format of the INPUT time for BCDBIN or the OUTPUT time for BINBCD.

**X12**

Indicates 12 hour format.

**X24**

Indicates 24-hour format.

X12 is the default if HOUR is omitted.

**REG=R|R2**

This is a general register that can be used by TYCVA. If omitted, the default is R2. (R0 is not valid for this parameter.)

**WORK=address**

The symbolic address of an 8-byte field aligned on a doubleword boundary.

**Entry Requirements**

None.

**Return Conditions**

- Control is returned to the next sequential instruction.
- The contents of the register specified in REG are unknown. The contents of all other registers are preserved across this macro call.
- The converted time will be placed in the field referenced by OUTPUT.
- When converting EBCDIC to binary, an alpha character other than P or M is treated as A (AM).

**Programming Considerations**

- This macro can be executed on any I-stream.
- The INPUT field is not altered at macro completion.
- The OUTPUT field will contain the converted time at macro completion.
- WORK and REG are used as work areas by TYCVA. Their contents are unknown at macro completion.

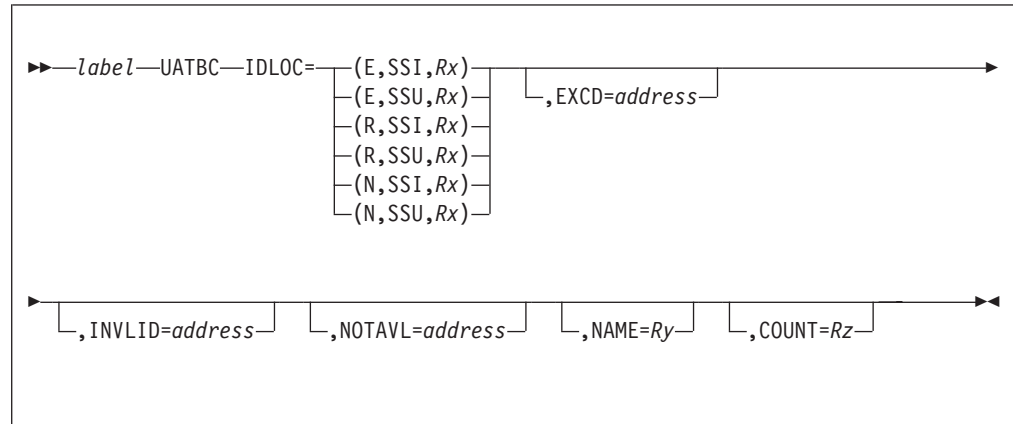
**Examples**

```
ANYNAME TYCVA MODE=BCDBIN,INPUT=EBW010,OUTPUT=EBW015,
 WORK=EBW024,REG=R3,HOUR=X12
```

## UATBC–MDBF User Attribute Reference Request

This general macro provides addressability to an SSU slot in the Subsystem User Table (MSOUT) for both C-type and E-type programs. The macro determines the requested SSU slot address and places it in a user-specified register so that upon return, the user can access the data in the SSUT via the MSOUT macro.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **IDLOC**

This is a required parameter that specifies the location of the SS/SSU ID to be used as input to the UATBC macro. Upon return, the general register specified by 'Rx' will contain the SSUT slot address of the specified SSU, adjusted for the SSUT header.

The valid request types are listed below.

#### **(E,SSI,Rx)**

This specifies that the input SS ID is supplied in the ECB field CE1DBI.

#### **(E,SSU,Rx)**

This specifies that the input SSU ID is supplied in the ECB field CE1SSU.

#### **(R,SSI,Rx)**

This specifies that the input SS ordinal number is supplied in the rightmost byte of 'Rx'. The high-order 3 bytes of 'Rx' are irrelevant.

#### **(R,SSU,Rx)**

This specifies that the input SSU ordinal number is supplied in the rightmost byte of 'Rx'. The high-order 3 bytes of 'Rx' are irrelevant.

#### **(N,SSI,Rx)**

This specifies that the input SS name is supplied in 'Rx'. The SS name must be a maximum of 4 characters, left justified, and padded with blanks on the right.

#### **(N,SSU,Rx)**

This specifies that the input SSU name is supplied in 'Rx'. The SSU name must be a maximum of 4 characters, left justified, and padded with blanks on the right.

#### **EXCD=address**

This is the address of a user exit routine to which control will be passed if the

'exceeded' condition is raised. This parameter is optional, but if omitted and the condition is raised, a system error is taken. This parameter is invalid if used with IDLOC=(N...).

**INVLID=address**

This is the address of a user exit routine to which control will be passed if the 'invalid' condition is raised. This parameter is optional, but if omitted and the condition is raised, a system error is taken. This parameter is ignored when IDLOC=(R...).

**NOTAVL=address**

This is the address of a user exit routine to which control will be passed if the 'not available' condition is raised. This parameter is optional, but if omitted and the condition is raised, a system error is taken.

**NAME=Ry**

This is the general register that is to contain the SS name, left justified, upon return. This SS is the parent SS of the SSU whose adjusted slot address is being returned by the UATBC macro. This parameter is optional.

**COUNT=Rz**

This is the general register that is to contain a number of SSUs upon return. If IDLOC=(x,SSI,x) is coded, this number is the total number of SSUs in the specified SS. If IDLOC=(x,SSU,x) is coded, this number is the number of SSUs remaining in the parent SS of the specified SSU, including the specified SSU itself. This parameter is optional.

## Entry Requirements

- If called by an E-type program, then R9 must contain the address of the ECB being processed.
- If called by a C-type program and IDLOC=(E,x,x) is specified, then R9 must contain the address of the ECB being processed.
- The location of the input to this macro is specified on the IDLOC parameter.

## Return Conditions

- Control is returned to the next sequential instruction.
- For E-TYPE programs the contents of R14 and R15 are unknown.
- For C-TYPE programs the contents of R14 are unknown.
- The contents of all other registers, excluding those specified in the macro parameters, are preserved across this macro call.
- The register specified in the IDLOC parameter contains the adjusted SSU slot address of an SSU within MS0UT.
  - If IDLOC=(x,SSI,x) is coded, the adjusted slot address of the first SSU within the specified SS is returned.
  - If IDLOC=(x,SSU,x) is coded, the adjusted slot address of the specified SSU is returned.
- The MS0UT DSECT tags should be used to reference the SSU slot returned, since there is no guarantee that the SSUT format will be unchanged in the future.
- If the NAME option is specified, the name of the parent SS of the SSU whose adjusted slot address is being returned by this macro is returned in the specified register, left justified, and padded with blanks on the right. The NAME option supersedes the IDLOC parameter. That is, if the same register is specified in both the IDLOC and NAME parameters, the contents of the register upon return will be a NAME, not an SSUT slot address.

## UATBC

- If the COUNT option is specified, a count is returned in the specified register.
  - If IDLOC=(x,SSI,x) is coded, this count is the total number of SSUs in the specified SS.
  - If IDLOC=(x,SSU,x) is coded, this count is the number of SSUs remaining in the parent SS of the specified SSU, including the specified SSU itself.

The COUNT option supersedes the NAME and IDLOC parameters. That is, if the same register is specified for the COUNT option as is specified for either the NAME or IDLOC parameters, the contents of the register upon return from the macro will be a count, not a NAME or an SSUT slot address.

- If an 'EXCD', 'INVLD', or 'NOTAVL' error condition is encountered while servicing a UATBC macro call, a branch is taken to the corresponding user exit label specified on the macro call, if provided. If one of these conditions is raised for which no user exit is provided, a system error is taken.
  - For E-type programs, the ECB is exited.
  - For C-type programs, the error is catastrophic.

The following table summarizes the circumstances that can raise the user error conditions:

| IDLOC |     | EXCD | INVLD | NOTAVL |
|-------|-----|------|-------|--------|
| E     | SSI | E1   | I1    | 01     |
| E     | SSU | E1   | I1    | 02     |
| R     | SSI | E2   | N/A   | 01     |
| R     | SSU | E2   | N/A   | 02     |
| N     | SSI | N/A  | I3    | 01     |
| N     | SSU | N/A  | I3    | 02     |

- E1** CE1DBI or CE1SSU complies with the identification integrity; that is, ordinal number plus complement is equal to X'FF'. However, the SS or SSU ordinal number exceeds the number of SSs/SSUs included in the last IPL.
- E2** The SS or SSU ordinal number supplied in 'Rx' exceeds the number of SSs/SSUs included in the last IPL.
- N/A** The condition does not occur.
- I1** CE1DBI or CE1SSU failed the integrity check—that is, ordinal number plus complement not equal to X'FF'.
- I3** The SS or SSU name supplied in 'Rx' cannot be found in subsystem attribute table or the subsystem user table, respectively.
- 01** The specified SS is indicated as inactive in CTKM.
- 02** The parent SS of the specified SSU is indicated as inactive in CTKM, or, the parent SS of the specified SSU is active but the specified SSU is indicated as dormant in MS0UT.
- System errors can result from the following conditions:
    - Invalid MDBF ID - The 'INVLD' error condition occurred.
    - Maximum MDBF ID exceeded - The 'EXCD' error condition occurred.
    - MDBF ID not available- The 'NOTAVL' error condition occurred.

- UATBC CP register save area in use - The UATBC macro was issued from a C-Type segment when the UATBC CP register save area was in use. Only one UATBC macro issued from a C-Type segment can be serviced at time.

## Programming Considerations

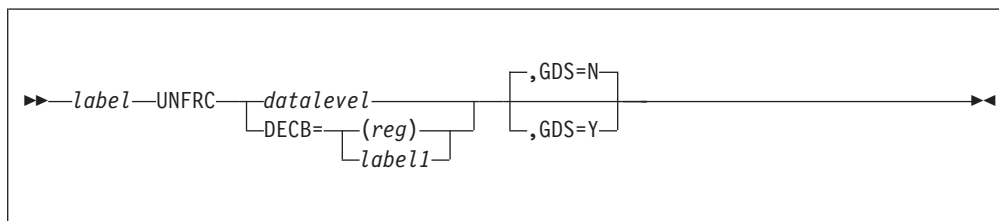
- This macro can be executed on any I-stream.
- The UATBC macro requires a minimum of eight bytes of storage when issued by E-type programs and a minimum of ten bytes of storage when issued by C-type programs. Four additional bytes of storage are required for each user exit coded.
- The UATBC macro expansion is independent of the operating environment. However, the macro service routine provided for a system generated with the MDBF option is different from that provided for a system generated without the MDBF option.
- Valid registers for the macro parameters are listed below:
  - E-type programs  
R0–R7
  - C-type programs  
R0-R8, and R9 if IDLOC=(E,x,x) is not coded. That is, R9 is valid for C-Type programs only if it does not contain an ECB address.
- The sequence of processing keywords that contain a register specification is IDLOC, NAME, COUNT. If the same register was specified for more than one of these keywords, control will return from the macro with the register containing data from the *last* keyword processed. For example, if R6 is specified for both IDLOC and NAME, upon return from the macro, R6 will contain a SS name and not an SSUT slot address.

## Examples

None.

This general macro releases the hold on the record address contained in the specified file address reference word (FARW) in the entry control block (ECB).

## Format



*label*

A symbolic name can be assigned to the macro statement.

datalevel

An ECB data level that identifies the file address and core block information for the I/O request.

**DECB=(reg)|label1**

The label or general register (R0–R7) containing the address of the data event control block (DECB), which specifies the file address and core block information for the I/O request.

## GDS

Specify one of the following:

**N** The file address in the specified ECB data level or DECB is for a record from the online database. N is the default.

**Y** The file address in the specified ECB data level or DECB is for a record on either a general file or a general data set.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- A file address that was previously held must be contained in the FARW for the specified ECB data level or DECB.
- The GDSRC macro must be executed to set up the file address before UNFRC is used to access a General Data Set record.

## 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 FARW at the specified ECB data level or DECB is unchanged.
- The file address contained in the specified FARW has been unheld. If another request was waiting to hold the same record, it has been serviced.
- If the UNFRC macro is issued outside a commit scope in a system that is not loosely coupled, the lock (hold) on the file address is immediately released. The lock can then be granted by the system to the next waiting ECB, if any.
- If the UNFRC macro is issued outside a commit scope in a loosely coupled system, the lock on the file address is released when all previous DASD writes

have been written to the media requested (virtual file access (VFA), DASD control unit cache, or the DASD surface). When the lock is released, the lock can then be granted by the system to the next waiting ECB, if any.

- If the UNFRC macro is issued inside a commit scope (that is, after a TXBGC macro call is issued and the commit scope has not been ended by a TXCMC or TXRBC macro call), the lock on the file address is not released to the system. Instead, the lock becomes available for use by any ECB operating in the same or lower-level commit scope.

When the commit scope ends with a TXCMC macro call, all pending DASD writes are written to the media requested (VFA, DASD control unit cache, or the DASD surface). After the DASD writes are completed successfully, locks are released to the system. At this point the locks can be granted to other ECBs that are not in the commit scope.

When the commit scope ends with a TXRBC macro call, all pending DASD writes are discarded. The UNFRC macro works differently depending on whether the lock was acquired as part of the current commit scope or before the current commit scope. If the lock was acquired before the current commit scope, the UNFRC macro is ignored on a rollback. This leaves the lock held by the ECB, perhaps as part of a previous commit scope. If the lock was acquired during the current commit scope, both the acquiring of the lock and its release are rolled back and the lock is immediately available. If the commit scope being rolled back is nested inside a higher-level commit scope, the lock may be released to either the TPF system or to the higher-level commit scope depending on whether the lock was previously part of the higher-level commit scope.

- A UNFRC macro cannot be issued on a record that is part of a suspended commit scope of an ECB. The following sequence will cause a system error:
  1. TXBGC
  2. FIWHC record x
  3. TXSPC
  4. UNFRC record x.

## Programming Considerations

- This macro can be run on any I-stream.
- A check is made to determine if the requested record is being presently held by this ECB. If the record is not presently held, control is transferred to the system error routine.
- 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.

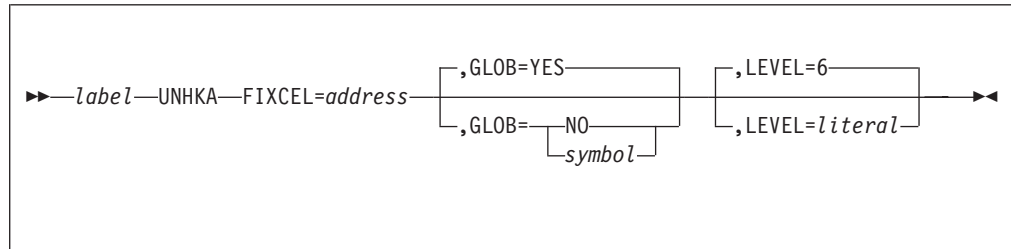
## Examples

None.

## UNHKA–Unhook Core Block

Use this general macro to transfer information from a requested core block reference word (CBRW) for a common block to a specified 8-byte field and detach (unhook) the common storage block associated with the CBRW. When a block has been unhooked in this manner, any other ECB can rehook the same block by using the REHKA macro. Blocks unhooked must be common blocks. These macros are designed for programs with unique buffering requirements.

### Format



*label*

A symbolic name can be assigned to the macro statement.

**FIXCEL=***address*

The operand address must be the symbolic address of the 8-byte field that will be used to save the requested CBRW.

**GLOB**

This is an optional keyword.

**YES**

This means the specified FIXCEL area is in global area 1. This is the default if GLOB is not coded.

**NO**

This means the specified FIXCEL area is not in protected application or global storage.

*symbol*

The operand symbol must represent a protected application or global storage area corresponding to the FIXCEL area location. Valid symbols are GLOBAL1, GLOBAL2, GLOBAL3, and APL1.

**LEVEL=***literal*

The operand literal must be a valid ECB data level ( $0 \leq \text{literal} \leq F$ ). If it is not coded, level 6 is assumed.

### Entry Requirements

- R9 must contain the ECB address.
- FIXCEL must reference a field of 8 bytes.
- A common storage block must be held in the referenced data level.
- The global fields must be defined and have a valid base if the GLOB symbol is a global.

### Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R0–R7 are preserved across this macro call.



- The contents of the core block reference word (CBRW) will be in the field referenced by the FIXCEL parameter.
- The CBRW will be initialized to indicate that a block is no longer attached at that level.
- The condition code is unchanged by UNHKA.

## Programming Considerations

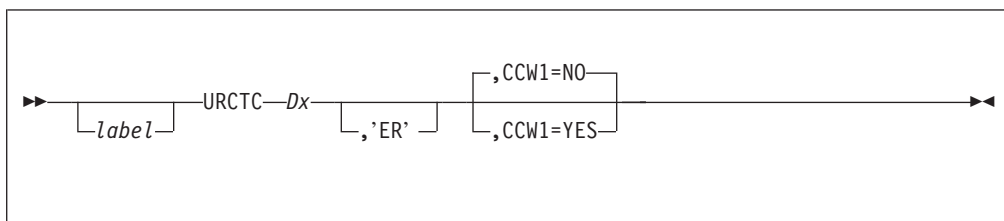
- This macro can be executed on any I-stream.
- This macro is for use by E-type programs only.
- The field defined by the FIXCEL parameter contains the contents of the CBRW specified by CE1CR(literal). It must be specified.
- The CBRW is initialized to indicate that a block is no longer held at that level.
- If the specified 8-byte field is in protected application or global storage, the GLMOD and KEYRC macros are used to allow modification of the specified field and restoration of the working storage protect key.
- Use of this macro must be limited to those programs having unique buffering requirements. It should not be used to obtain an additional data level.
- Any application program using the UNHKA-REHKA facility must be designed so that 2 successive UNHKA macros referencing a given field are always separated by a REHKA referencing the field.
- UNHKA uses the GLMOD and KEYRC macros.

## Examples

None.

This general macro causes user-provided CCWs to be run to a specified unit record device. It is used to perform special functions such as buffer loading and error recovery. It should be used only to perform system utility functions. It should not be used by application programs.

## Format



*label*

A symbolic name may be assigned to the macro statement.

**Dx** A file address reference word (D0–DF) must be specified.

ER

Specifies the error recovery restart function.

CCW1

This parameter is used to indicate the CCW format in the specified data level.

**YES**

CCWs are treated as format-1 CCWs.

**NO**

CCWs are treated as format-0 CCWs.

The default assignment is NO.

## Entry Requirements

- R9 must contain the address of the ECB being processed.
- The file address reference word for the level specified contains:

When you do not specify the ER parameter:

**CE1FA(x)**      Address of the start of the CCWs.

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <b>CE1FM(x)</b> | The 4-byte physical device address. This address must be in the Unit Record Status Table. |
|-----------------|-------------------------------------------------------------------------------------------|

When you specify the ER parameter:

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <b>CE1FA(x)</b> | Address of parameter to be passed to the Unit Record Support CSECT. |
|-----------------|---------------------------------------------------------------------|

**CE1FM(x)**      The 4-byte physical device address.

## 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 contents of the remaining operational registers and the condition code are saved during the execution of this macro.
- When you do not specify the ER parameter, the status of the operation is unknown until the next executed WAITC macro.
- When you specify the ER parameter, no I/O is requested and consequently a WAITC macro is not needed. The requested function is performed by the unit record support CSECT and return is made to the next sequential instruction.

## Programming Considerations

- This macro may be executed on any I-stream.
- The ECB reference register (R9) must contain the address of the ECB issuing this macro.
- Because this macro is used to perform control functions, no USURC macro need be issued by the ECB. Unlike the other Unit Record I/O macros that service requests on a first-in-first-out basis, the URCTC request is placed at the top of the request queue.
- For a URCTC request to be serviced, one of the following conditions must be satisfied:
  - The device is not assigned to any ECB.
  - The device, although assigned to an ECB, is suspended because of hardware errors.
  - The device is assigned to the ECB issuing this URCTC request.

If none of the above conditions is met, the error return on the WAITC will be taken when the WAITC is issued.

- The supported CCW command codes are:

| Code  | For Device | Function Performed           |
|-------|------------|------------------------------|
| X'02' | 3211       | Read PLB                     |
| X'04' | 3211       | Sense                        |
| X'06' | 3211       | Check Read                   |
| X'0A' | 3211       | Read UCSB                    |
| X'12' | 3211       | Read FCB                     |
| X'23' | 3211       | Unfold                       |
| X'43' | 3211       | Fold                         |
| X'63' | 3211       | Load FCB                     |
| X'73' | 3211       | Block Data Check             |
| X'7B' | 3211       | Allow Data Check             |
| X'FB' | 3211       | Load UCSB                    |
| X'73' | 1403       | Block Data Check Latch       |
| X'7B' | 1403       | Reset Block Data Check Latch |
| X'EB' | 1403       | Allow Buffer Loading         |
| X'F3' | 1403       | Load UCSB (Folding)          |
| X'FB' | 1403       | Load UCSB (No Folding)       |

- When you specify the ER parameter, the parameter passed to the Control Program indicates in byte 0:

## URCTC

| Bit | Meaning when set to 1: |
|-----|------------------------|
|-----|------------------------|

|   |                         |
|---|-------------------------|
| 0 | Reissue the failing CCW |
|---|-------------------------|

|   |                                                                                                                                  |
|---|----------------------------------------------------------------------------------------------------------------------------------|
| 1 | Promote the device request queue and wait for an interrupt (such as the operator pressing the STOP, START button on the device). |
|---|----------------------------------------------------------------------------------------------------------------------------------|

|   |                                                                |
|---|----------------------------------------------------------------|
| 2 | Promote the device request queue and service the next request. |
|---|----------------------------------------------------------------|

|   |                |
|---|----------------|
| 3 | Abort the job. |
|---|----------------|

- When you do not specify the ER parameter, a WAIT macro must be issued to wait for the completion of the I/O operation. When the error return is taken, the CE1SUD for the specified level and CE1SUG will be set as follows:

| Bit | Meaning when set to 1: |
|-----|------------------------|
|-----|------------------------|

|   |                                                                                                                  |
|---|------------------------------------------------------------------------------------------------------------------|
| 4 | An entry error condition is detected or the given device address is not in the Unit Record Status Table (UR1ST). |
|---|------------------------------------------------------------------------------------------------------------------|

|   |                                                                                                                                                                                          |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5 | A software error, such as command reject or protection check indicator set in the CSW, has been detected, and the failing 8-byte CCW is moved into the CE1FA(X) for the level specified. |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|   |                                                     |
|---|-----------------------------------------------------|
| 6 | Device error encountered and the user should abort. |
|---|-----------------------------------------------------|

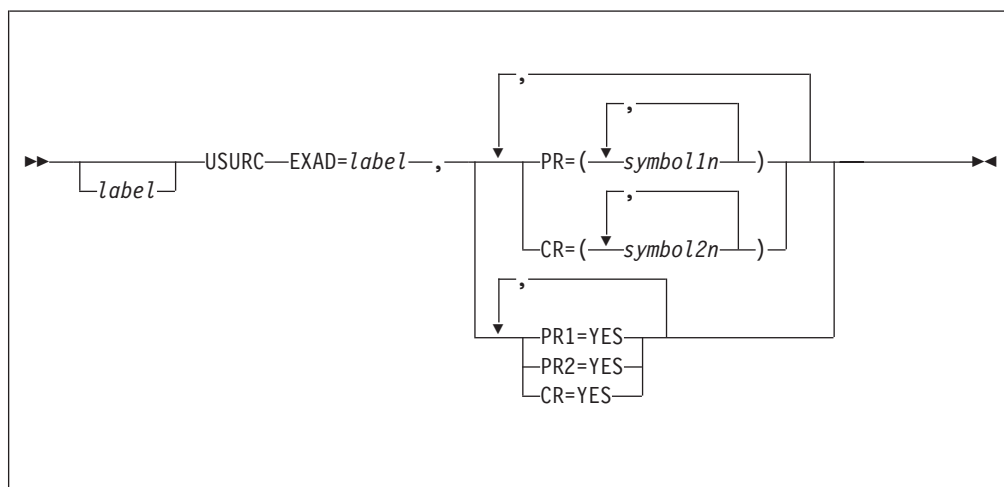
## Examples

None.

## USURC—Assign Unit Record Devices

This general macro assigns the requested unit record devices (card readers and printers) to the issuing application program.

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **EXAD=*label***

This is required. The label of an operational program error routine within the current program segment.

#### **PR=*symbol1n***

Each parameter specifies the symbolic number of a printer (1–7), or specifies the available printer (A). The only valid parameters are 1, 2, 3, 4, 5, 6, 7, and A. The PR parameter must be specified if CR is not specified. The available printer is the first printer defined to TPF.

#### **CR=*symbol2n***

Each parameter specifies the symbolic number of a card reader (1–3) or specifies the available card reader (A). The only valid parameters are 1, 2, 3, and A. The CR parameter must be specified if PR is not specified. The available card reader is the first card reader defined to TPF.

#### **PR1=YES**

This specifies that Printer Number One is to be used. YES is the only allowed value. One of PR1, PR2, and CR must be coded.

#### **PR2=YES**

Specifies that Printer Number Two is to be used. YES is the only allowed value. One of PR1, PR2, or CR must be coded.

#### **CR=YES**

This specifies that Card Reader One is to be used. YES is the only allowed value. One of PR1, PR2, or CR must be coded.

### Entry Requirements

R9 must contain the address of the entry control block (ECB).

## USURC

### Return Conditions

- If each specified device is available for assignment, control is returned to the next sequential instruction (NSI). Otherwise, control is returned to the address specified by EXAD.
- The contents of scratch registers R14 and R15 are unknown. The condition code is not saved across the execution of this macro. The contents of the remaining registers are saved.
- The specified devices are assigned for use by this program until the EXITC macro is issued.

### Programming Considerations

- A check is made by the control program to determine that the devices are assignable and this macro has not been previously issued by this application program.
- All unit record devices to be used by the application program must be specified at one time. This means that this macro must be issued only once by a program. It must be issued before any unit record I/O macro is issued.
- A condition code of zero when the macro returns indicates all devices specified are assigned to this ECB and any unit record I/O macro (PRLNC, RDCDC) can be issued.

If the error return address is taken, none of the devices has been assigned to this ECB. Bit 7 of CE1URM in the ECB is set to 1 if all of the requested devices can be assigned to this ECB.

- This macro checks only for the availability of the devices and does not check whether the device is ready or operational. These conditions will be detected when a unit record I/O macro (PRLNC, RDCDC) is run.
- The USURC macro may be executed on any I-stream.

### Examples

These two examples are equivalent:

```
USURC EXAD=ERROR,PR=(1,2),CR=1
USURC EXAD=ERROR,PR1=YES,PR2=YES,CR=YES
```

# UXCMC–User Exit Control

This general macro is used to dynamically activate or deactivate one or more of the dynamic user exit points in the control program. The UXCMC macro is also used to activate or deactivate one or more functions that reside in dynamic or nondynamic user exits. Optionally, when you use this macro, you can specify an address (override address) of a user exit routine that will receive control from the specified exit point.

## Format



*label*  
A symbolic name can be assigned to the macro statement.

**WKREG=*Rx***  
WKREG identifies a register to be used when UXCMC is called from the CP. The WKREG parameter should not be used by E-type programs. It is required when called from the CP. The WKREG must be R0, R2–R7. The original contents of the specified register are not saved.

## Entry Requirements

- R1 must contain the address of the parameter list. There is no boundary requirement for the parameter list. However, processing is more efficient if the parameter list begins on a fullword boundary (see the UX1PL data macro for a description of the parameter list).
- The format of the parameter list is:

| Byte  | Description                                                                                                                                                                                           |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0     | Action Code<br><br>C"A"    Activate specified exit points<br>C"D"    Deactivate specified exit points                                                                                                 |
| 1     | Parameter List Type<br><br>C"N"    New list format<br>X'??'    Anything other than 'N' implies an old list format                                                                                     |
| 2–3   | Number (hexadecimal value) of exit points being activated or deactivated                                                                                                                              |
| 4–5   | Exit point index, obtained from UCL (hexadecimal value)                                                                                                                                               |
| 6–7   | When the exit point index is either CLE (library function entry) or CLX (exit user exit), this field must contain a library ordinal between 0–1023 or it must contain 1024 to indicate all libraries. |
| 8–11  | Zeros or an override address (valid only on activation). If an override address is specified, it must be a valid storage address of a user exit routine.                                              |
| 12–19 | If multiple functions are defined, this field will contain the bit settings for the functions to be activated or deactivated for the specified exit point.                                            |

For the previous parameter list format, bytes 4–11 are repeated for each exit point; for the new format, bytes 4–19 are repeated.

## Return Conditions

- Control is returned to the next sequential instruction.
- The contents of R0–R7 are preserved across this macro call.
- The contents of R14 and R15 are described below.

R15 contains a return code as specified below. R14 is set depending on the return code. If the return code is nonzero, the exit points have **not** been acted upon.

| Contents of R15 | Contents of R14                    | Meaning                                                                                                                                                              |
|-----------------|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0               | 0                                  | Normal Return. The specified exit points have been activated or deactivated.                                                                                         |
| 4               | 0                                  | Incorrect action specified.                                                                                                                                          |
| 8               | 0                                  | Too many exit points specified in the parameter list. The number exceeds the maximum defined in the DCTUCL DSECT (UCLCNT).                                           |
| 12              | address of incorrect entry         | Incorrect index specified in the parameter list. The index is out of range.                                                                                          |
| 16              | address of incorrect entry         | An exit point index was specified in the parameter list for a nondynamic exit point and multiple function support is not being used.                                 |
| 20              | address of duplicate entry         | A duplicate index value was specified in the parameter list.                                                                                                         |
| 24              | address of incorrect entry         | An override address and activate action code were specified but the exit point was already active or multiple overrides were specified on ISO-C library exits.       |
| 28              | address of incorrect entry         | The parameter list specifies an exit point for activation that has no user exit routine supplied for it. The routine must be coded and assembled in CP CSECT CCUEXT. |
| 32              | The address of the incorrect entry | The parameter list specifies either CLE or CLX as exit points but the library ordinal number coded is greater than the maximum (1024).                               |

## Programming Considerations

- This macro can be executed on any I-stream.
- If called from the control program, R13 must point to a stack.
- See *TPF System Installation Support Reference* for additional information concerning user exits.
- UX1PL is the DSECT for the parameter list. The UX1PL macro does not have any input parameters and does not generate a USING statement.
- Only dynamic exit points can be specified in the parameter list for activation or deactivation. Nondynamic exit points can only be specified in the parameter list when activating or deactivating functions.
- The exit point index may be obtained by searching the UCL based on the exit point name. CINFC label CMMUCL provides addressability to the UCL.



- The library ordinal for an ISO-C library is found by searching the Shared Library Names Table (SLNT) using the library name. The CINFC label CMMICD provides addressability to the data mapped by the IDSICD DSECT which contains the SLNT address.
- If activate is specified and an exit point and any specified functions are already active, no processing for the exit point is performed; this is **not** treated as an error condition.
- You can override the address of the user exit routine only if the exit point is inactive. It is an error if the exit point is already active and an override address is specified. The original user exit address for an exit point is restored when the exit point is deactivated.
- A deactivate request will turn off the specified function bits and will only set the exit as inactive when all of the function bits have been turned off.
- If deactivate is specified and an exit point is already inactive, no processing for the exit point is performed; this is **not** treated as an error condition.
- User exit routines must be capable of receiving control as soon as this macro activates an exit point. For example, if the SVC macro exit (SVX) is activated, the user exit routine is invoked at the end of the macro processing for the same call the UXCMC macro prior to returning control to the program that ran the UXCMC macro.
- It is the **user's** responsibility to manage the activation and deactivation of exit points used for multiple user functions.
- An exit point is activated only if a user exit routine exists for the exit point. If the user exit routine has not been assembled in CCUEXT, its address must be specified in the parameter list, UX1PL (override address).
- When the exit point specified is either the library function entry (CLE) or the exit (CLX) user exit, the following rules apply:
  - There is only 1 CLE user exit routine and only 1 CLX user exit routine for all ISO-C libraries. If CLE or CLX has an override address specified for any ISO-C library, the override address applies to all ISO-C libraries for the user exit.
  - For CLE or CLX, if the parameter list contains entries for specific libraries as well as an entry for all libraries, the specific library entries are ignored and the entry for all libraries is processed. No error condition is returned.
  - For CLE or CLX, if an entry for all libraries is specified more than once, only the first such entry is processed. No error condition is returned.

## Examples

- In this example (previous format), two exit points will be activated, and the user exit routine address for the exit point whose index is 2 will be overridden in the UCL.

|    |  |      |      |  |          |      |  |          |
|----|--|------|------|--|----------|------|--|----------|
| A  |  | 0002 | 0004 |  | 00000000 | 0002 |  | 0034FDE0 |
| 00 |  | 02   | 04   |  | 08       | 0C   |  | 14       |

- In this example (new format), two exit points are to be activated along with their associated functions. The stub routine for exit 1 resides in CUSR(CCUEXT). The

## UXCMC

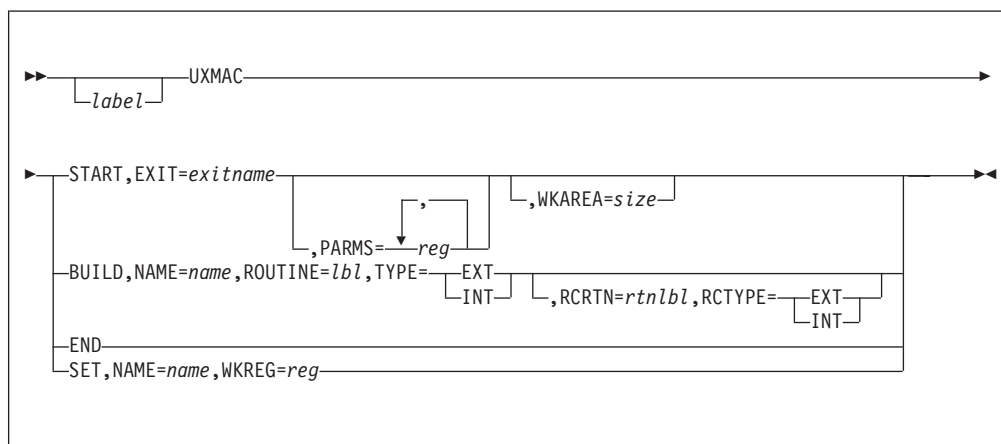
stub routine for exit 2 could reside elsewhere because its address has been overridden.

|    |    |      |      |    |          |                  |      |  |          |                  |
|----|----|------|------|----|----------|------------------|------|--|----------|------------------|
| A  | N  | 0002 | 0004 |    | 00000000 | 8400000000000000 | 0002 |  | 0034FDE0 | 1800000000000000 |
| 00 | 02 | 04   | 08   | 0C | 14       | 18               | 1C   |  |          |                  |

## UXMAC—Define Multiple User Exit Interface

This general macro defines and generates the code and tables that are necessary to support the use of a user exit by multiple functions.

### Format



#### START

Defines the start of the multiple function user exit interface build process.

#### EXIT=*exitname*

Specifies the name of the exit for which this build process is being executed, where *exitname* is the 1- to 4- character name defined in DCTUCL.

#### PARMS=*reg*

Specifies the register (or registers) that are not to be reset from the stack because they contain data. When more than one register is specified, all the registers must be in parentheses.

#### WKAREA=*size*

Specifies, in bytes, a user work area to be allocated on the current stack. A label to access this area is generated in the form *name\_WKA*, using the name specified with the EXIT parameter.

#### BUILD

Defines the information associated with a single function. The order of the UXMAC macro statements with the BUILD parameter specified dictates the order in which the functions will receive control when active.

#### NAME=*name*

Specifies the name of the requested function, where *name* is one of the equate names defined in IUXEQ.

#### ROUTINE=*lbl*

Specifies the name of the routine (entry point) that will process the specified function.

#### TYPE

Specifies the type of addressability the stub routine has to the called routine identified by the ROUTINE parameter.

#### EXT

Specifies that the address of the called routine is located in a different CSECT than this routine. The address is resolved by a V-type constant (Vcon).

## UXMAC

### INT

Specifies that the address of the called routine is within the current CSECT. The address is resolved by an address constant (Adcon).

### RCRTN=*rtnlbl*

Specifies the name of the routine that will process a return code from the specified function.

### RCTYPE

Specifies the type of addressability the stub routine has to the called routine identified by the RCRTN parameter. This parameter is required when RCRTN is coded.

### END

Defines the end of the definition interface macros and initiates the generation of executable code and tables.

### SET

Turns on the multiple-function indicator in a parameter list entry.

### NAME=*name*

One of the equate names, defined in IUXEQ, of the function to be set on.

### WKREG=*wkreg*

Specifies an even-odd pair of registers to use as work registers. The even register must be specified.

## Entry Requirements

None.

## Return Conditions

None.

## Programming Considerations

- Inline code is generated.
- The START, BUILD, and END parameters are designed for use in the control program (CP) only and will generate the following set of items:
  - An address table for access to function processing routines
  - An address table for access to return code processing routines
  - A bit map table to define function bits and order of activation
  - Code to process the bit table and call the function and return code routines.
- The SET parameter can be used wherever a parameter list is constructed and will generate code to initialize a function bit in a parameter list entry.
- The stub code generated by this macro makes use of the SLNKC macro to preserve the calling information on the stack so it is not necessary to code the SLNKC macro in front of the UXMAC macros.
- In the normal scope of coding a multifunction user exit, it is only necessary to code a START, a BUILD for each function, and an END. No additional code is required, and each function will be called in the order of the BUILD statements.
- Sometimes it is necessary to pass information (for example, return codes) from one function to the next and from a function back to the routine calling the exit. This can be done by using the PARMS, WKAREA, and RCRTN parameters. In addition to using these parameters, it is possible to add code to the stub routine itself to provide for initialization and control of additional work areas. You can add

code in two locations, between the START statement and the first BUILD, and between the last BUILD and the END statement. See the coding example for more information.

## Examples

- The following example shows a user exit, called UEX1, that is defined with two functions: IUX\_UF1 and IUX\_UF2. The IUX\_UF1 function must be called first and sets up a return code to be passed back, in R2, to the routine calling the exit. Both functions require an additional work area.

```

UCCUEX1 DS 0H

 UXMAL START,EXIT=UEX1,PARMS=R2,WKAREA=12

 ST R2,UEX1_WKA Save callers R2
 $GSWBC BLOCK=R2,SAVREG=(R0,R1) Get additional work area
 ST R2,UEX1_WKA+4 Save the address of the SWB
 MVC 0(4,R2),ENTN_WKA Pass SWB and R2 value to functions
*
* Now call all of the functions defined for UEX1.
*
 UXMAL BUILD,NAME=IUX_UF1,ROUTINE=CPUF1,TYPE=INT,RCRTN=UF1_RC, X
 RCTYPE=INT
 UXMAL BUILD,NAME=IUX_UF2,ROUTINE=CPUF2,TYPE=EXT
*
* The functions have all been called, get rid of the SWB and pass the
* return code back to the caller.
*
 L R1,ENTN_WKA+4 Get address of SWB
 $RSWBC BLOCK=R1 Release SWB
 L R2,ENTN_WKA+8 Load return code from UF1_RC

 UXMAL END
 .
 .
 .
UF1_RC DS 0H Process return code
 USING *,R15
 SLNKC LOREG=NO,PUSH=NO
 ST R2,ENTN_WKA+8 Save return code
 RLNKC LOREG=NO,POP=NO
 LTORG
 DROP R15

```

The previous macro calls will generate the following stub routine:

```

UCCUEX1 DS 0H
+ USING UCCUEX1,R15 register for this routine
+ SLNKC PUSH=YES,DSECT=UEX1,LOREG=R0
+UEX1_INL14 DS F Save area for callers R14
+UEX1_UREGS DS 4F User register save area
+UEX1_CREGS DS 2F UXMAL register save area
+UEX1R2 DS F Save area for PARMS register
+UEX1_WKA DS XL12 User work area
+ DLNKC
+ MVC UEX1_INL14,UEX1LEN+STKINL14(R13) Save callers R14

 ST R2,UEX1_WKA Save callers R2
 $GSWBC BLOCK=R2,SAVREG=(R0,R1) Get additional work area
 ST R2,ENTN_WKA+4 Save the address of the SWB
 MVC 0(4,R2),ENTN_WKA Pass SWB and R2 value to functions

+ STM R0,R3,UEX1_UREGS Save user registers
+ L R2,=V(UCLFUNI) Load base of UCL fun inds
+ LA R3,UCLQUX1 Load exit point index
+ MH R3,=Y(L'UCLFUNI) Get fun ind. displacement

```

## UXMAC

```

+ AR R2,R3 Add to UCL fun base entry
+ L R3,CNT_UEX1 Load count of functions
+ MH R3,FSZ_UEX1+2 X size = displacement
+CK_UEX1 DS 0H
+ L R0,0(R2) Load multi-function bits
+ L R1,4(R2) Load multi-function bits
+ L R14,FUNI_UEX1+0(R3) Load bit position
+ SLDL R0,0(R14) Move bit to position 0-R0
+ LTR R0,R0 Is this function active?
+ BNM NXT_UEX1 No, go check next function
+ L R14,FUNA_UEX1+0(R3) Load routine address
+ STM R2,R3,UEX1_CREGS Save UXMAC control registers
+ LM R0,R3,UEX1_UREGS Restore user registers
+ MVC STKINL14(R13),UEX1_INL14 Set up callers R14 for exit
+ CLNKC BASE=R15,LINK=R14,RTN=(R14) Call user function
+ L R14,UEX1_CREGS+4 Load function pointer
+ L R14,FUNR_UEX1+0(R14) Load addr of return code routine
+ LTR R14,R14 Is there a return code routine?
+ BZ SK_UEX1 No, continue
+ CLNKC BASE=R15,LINK=R14,RTN=(R14) Call return code routine
+SK_UEX1 DS 0H
+ LM R2,R3,UEX1_CREGS Restore UXMAC work registers
+NXT_UEX1 DS 0H
+ SL R3,FSZ_UEX1 Point to next function
+ BNM CK_UEX1 Go if more functions to check

 L R1,UEX1_WKA+4 Get address of SWB
$RSWBC BLOCK=R1 Release SWB
 L R2,UEX1_WKA+8 Load return code from UF1_RC

+ RLNKC POP=YES,LOREG=R0,PARMS=R2
++
++
++ +CNT_UEX1 DC F'1' Count of functions minus 1
++ +FSZ_UEX1 DC F'12' Function table entry size
++
++ +FUNT_UEX1 DS 0F Function table
++ +FUNA_UEX1 EQU FUNT_UEX1,4 Function address
++ +FUNR_UEX1 EQU FUNA_UEX1+4,4 Return code routine address
++ +FUNI_UEX1 EQU FUNR_UEX1+4,4 Function indicator
++
++ Each entry in the function table is three words in length. The
++ first word contains the address of the routine that services the
++ function. The second word contains the address of the routine
++ that services the return code returned from the function. The
++ third word contains the function indicator bit.
++
++ + DC V(CPUF2) Address of function
++ + DC A(0)
++ + DC A(IUX_UF2) Function indicator bit
++
++ + DC A(CPUF1) Address of function
++ + DC A(UF1_RC) Address of return code routine
++ + DC A(IUX_UF1) Function indicator bit

```

- The following code example builds the parameter list required to activate the UEX1 user exit along with its associated functions.

```

.
.
DCTUCL CALL UCL EQUATES
UX1PL DSECT FOR PARAMETER LIST
LA R1,CT99_WORK ADDRESS UX1PL WORK AREA
USING UX1PL,R1
MVI UX1COD,UX1ACT SET ACTIVATE REQUEST
MVI UX1PLF,UX1NEW SET NEW LIST FORMAT
MVC UX1CNT,=H'2' SET COUNT OF LIST ENTRIES
MVC UX1IDX,=AL2(UCLEQUEX1) SET INDEX FOR 'UEX1' EXIT

```

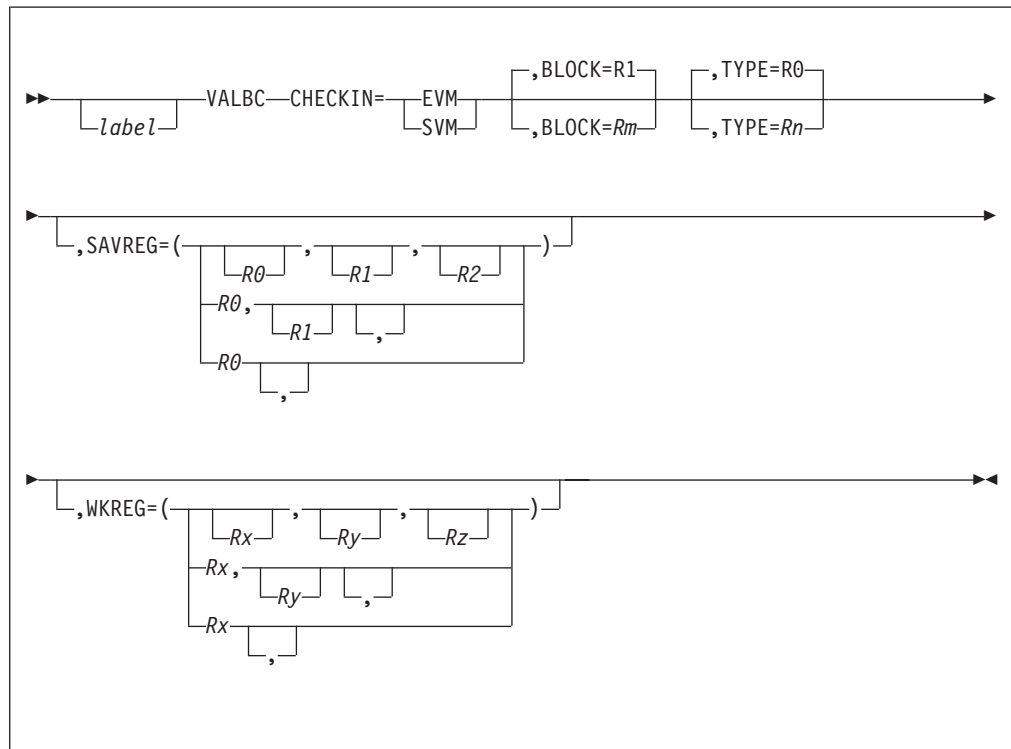
## UXMAC

```
UXMAC SET,NAME=IUX_UF1,WKREG=R2 SET UF1 FUNCTION ON
UXMAC SET,NAME=IUX_UF2,WKREG=R2 SET UF2 FUNCTION ON
UXCMC WKREG=R2 ACTIVATE 'UEX1' EXIT AND FUNCTIONS
.
.
```

## VALBC—Validate Storage Block Address

Use this general macro to validate a storage block address and determine whether the block is in use.

### Format



#### CHECKIN

Specifies the address space containing the address specified in the BLOCK parameter. The CHECKIN parameter is required and one of these address spaces must be specified.

#### EVM

When specified, the block address is checked for validity in the EVM and is considered valid only if the address is within the ECB's virtual memory (EVM).

#### SVM

When specified, the block address is checked for validity in the SVM and is considered valid only if the address is within the system's virtual memory (SVM).

#### BLOCK=R1|Rm

This parameter indicates the block to be validated. The register specified contains the address of the storage block to be validated. This address can be either an SVA or an EVA address, but will be validated depending on the setting of the CHECKIN parameter.

The default assignment is R1.

#### TYPE=R0|Rn

The register specified in this parameter contains the storage block type. The default assignment is R0. The following types are supported:



**L0** 128 byte block

**L1** 381 byte block

**L2** 1055 byte block

**L4** 4095 byte block

**LECB (L3)**

Entry Control Block

**LSWB**

System Work Block

**LIOCB**

I/O Control 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.*

## Entry Requirements

- The register specified in the TYPE parameter contains a valid block type.
- The register specified in the BLOCK parameter contains the block address.

## Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown.
- The TYPE register will contain the block type.
- Condition codes returned:

The condition code is set to “0”, when the storage block address provided is valid and the block is in use.

**Note:** When IOBs are validated, there is no check to validate that the block is in use.

The condition code is set to “1”, when the storage block address provided is not in use, but the address exists in the ECB’s address space.

The condition code is set to “2” in any of the following situations:

- For physical (LIOCB, LSWB or LECB) types:
  - The storage block address does not fall in the range of addresses for blocks of the type specified in the TYPE parameter.
  - The storage block address does not fall on a block boundary.

## VALBC

- For logical (L0, L1, L2, or L4) types:
  - The storage block address can not be resolved in the ECB's address space.
  - The record type associated with the storage block address does not match the record type specified in the TYPE parameter.

**Note:** The condition code is set differently than in prior releases. Previously a condition code of '1' was always returned when a block was released. In the current release either condition code '1' or condition code '2' will be returned after a block is released. Condition code '2' is returned, because when virtual storage management frees the storage backing the released block, the block address for the ECB becomes invalid.

## Programming Considerations

- This macro may be executed on any I-stream.
- This macro has a supervisor call (SVC) interface for use by E-type programs.
- If this macro is called from the control program, the caller must be in supervisor state.

## Examples

None.



## VCHKC

|        |                                               |                                                                                          |
|--------|-----------------------------------------------|------------------------------------------------------------------------------------------|
| BNZ    | ERROR                                         | Process Errors                                                                           |
| LR     | R6,R14                                        | Save ACB pointer in R6                                                                   |
|        |                                               |                                                                                          |
| VOPNC  | ACB=(R6)                                      | Open file                                                                                |
| BNZ    | ERROR                                         | Process Errors                                                                           |
|        |                                               |                                                                                          |
| VGENC  | BLK=RPL,<br>AM=VSAM,<br>ACB=(R6),<br>LEVEL=D1 | Generate an RPL<br>..Access method is VSAM<br>..ACB pointer in R6<br>..Use data level D1 |
|        |                                               |                                                                                          |
| VGETC  | RPL=(R7)                                      | Retrieve a record                                                                        |
| BNZ    | ERROR                                         | Process Errors                                                                           |
|        |                                               |                                                                                          |
| VCHKC  | RPL=(R7)                                      | Wait for I/O completion                                                                  |
| BNZ    | ERROR                                         | Process Errors                                                                           |
|        |                                               |                                                                                          |
| .      |                                               |                                                                                          |
| .      |                                               |                                                                                          |
| .      |                                               |                                                                                          |
| DDNAME | DC CL8'TRANSLOG'                              | Data Definition Name                                                                     |



## VCLSC

|        |                  |                         |
|--------|------------------|-------------------------|
| VOPNC  | ACB=(R6)         | Open file               |
| BNZ    | ERROR            | Process Errors          |
| VGENC  | BLK=RPL,         | Generate an RPL         |
|        | AM=VSAM,         | ..Access method is VSAM |
|        | ACB=(R6),        | ..ACB pointer in R6     |
|        | LEVEL=D1         | ..Use data level D1     |
| VGETC  | RPL=(R7)         | Retrieve a record       |
| BNZ    | ERROR            | Process Errors          |
| VENDC  | RPL=(R7)         | End the request         |
| BNZ    | ERROR            | Process Errors          |
| VCLSC  | ACB=(R6)         | Close the file          |
| BNZ    | ERROR            | Process Errors          |
| .      |                  |                         |
| .      |                  |                         |
| .      |                  |                         |
| DDNAME | DC CL8'TRANSLOG' | Data Definition Name    |



## VENDC

|        |           |               |                         |
|--------|-----------|---------------|-------------------------|
| VGENC  | BLK=RPL,  |               | Generate an RPL         |
|        | AM=VSAM,  |               | ..Access method is VSAM |
|        | ACB=(R6), |               | ..ACB pointer in R6     |
|        | LEVEL=D1  |               | ..Use data level D1     |
| VGETC  | RPL=(R7)  |               | Retrieve a record       |
| BNZ    | ERROR     |               | Process Errors          |
| VENDC  | RPL=(R7)  |               | End this request        |
| BNZ    | ERROR     |               | Process Errors          |
| VCLSC  | ACB=(R6)  |               | Close the file          |
| BNZ    | ERROR     |               | Process Errors          |
| .      |           |               |                         |
| .      |           |               |                         |
| .      |           |               |                         |
| DDNAME | DC        | CL8'TRANSLOG' | Data Definition Name    |



---

## VGENC—Generate an Access Method Control Block or Request Parameter List

Use this general macro to generate the following:

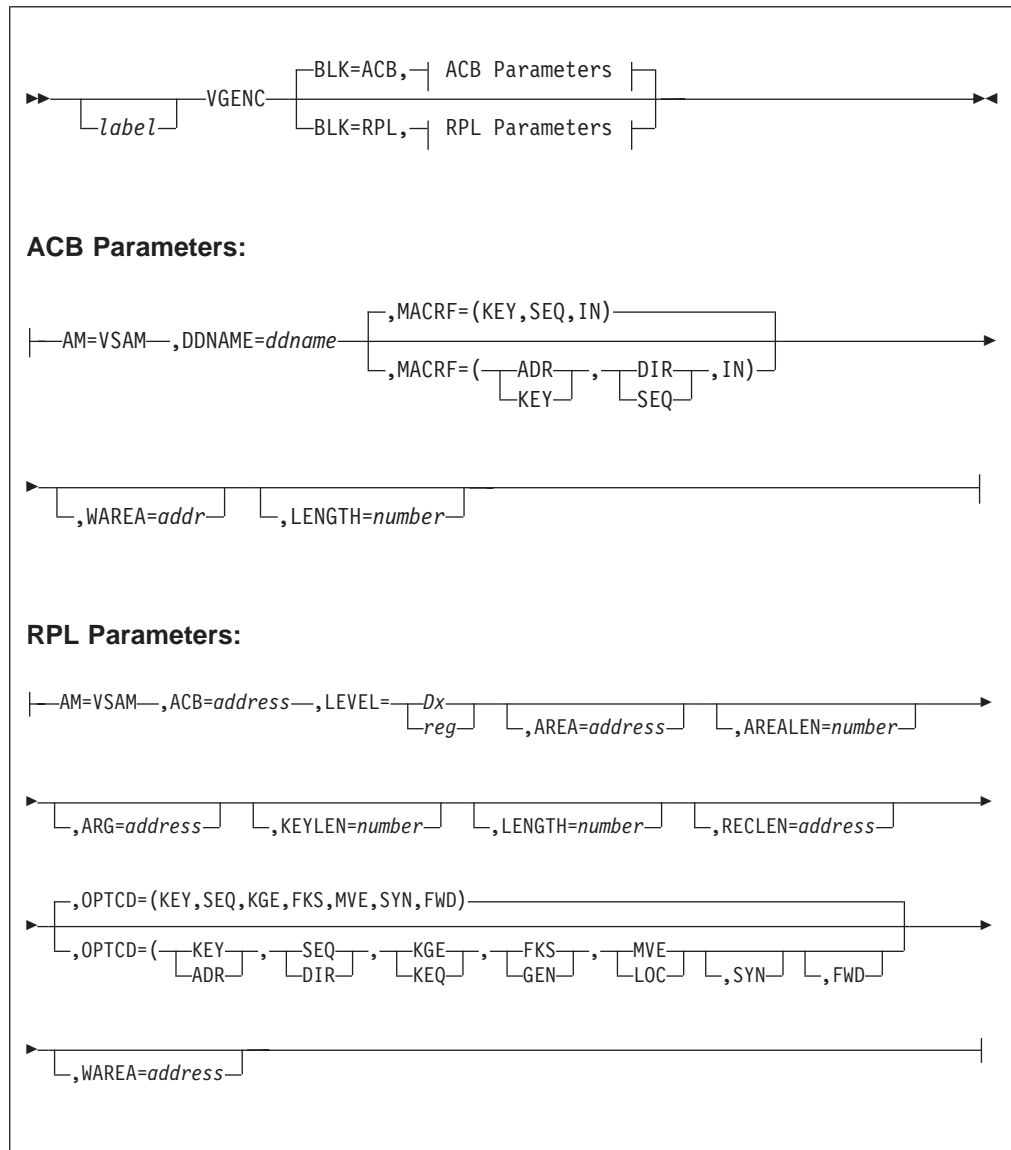
- An access method control block (ACB)

The ACB is used to access a virtual storage access method (VSAM) data set. It can be generated in automatic storage that is maintained by VSAM database support on data level DF or in a user area by coding the WAREA parameter.

- A request parameter list (RPL)

The RPL is used to control an active request on a VSAM data set. Storage for the RPL can be generated in automatic storage that is maintained by VSAM database support on data level DF or in a user area by coding the WAREA parameter.

## VGENC Format



### *label*

A symbolic name can be assigned to the macro statement.

### **BLK=ACB**

Specifies that an ACB will be generated.

### **AM=VSAM**

Specifies that the access method using this control block is VSAM.

### **DDNAME=ddname**

Specifies an 8-character name that identifies the VSAM cluster you want to process. The TPF system appends 4 characters to the data definition (DD) name, as follows:

- The first 2 characters indicate the index (IN) or data (DA) portion of the VSAM clusters.
- The second 2 characters indicate the prime (PR) or mirror (MI) portion of the VSAM clusters.

**MACRF**

Specifies processing options as follows:

**ADR**

Specifies the address access to a key-sequenced or entry-sequenced data set. Relative byte addresses (RBAs) are used as search arguments. Sequential access is by entry sequence.

**KEY**

Specifies keyed access to a key-sequenced data set. Keys or RBAs are used as search arguments. Sequential access is by key or RBA.

**DIR**

Specifies direct access to a key-sequenced or entry-sequenced data set.

**SEQ**

Specifies sequential access to a key-sequenced or entry-sequenced data set.

**IN** Open the data set for input.

**WAREA**

Specifies the address of an area for the ACB or RPL to be generated.

**Note:** If you do not specify an area for the ACB or RPL to be generated, VSAM database support gets working storage for the area.

**LENGTH**

Specifies the length, in bytes, of the area that you are designating (in the WAREA parameter) for VSAM database support to generate the ACB or RPL. The maximum value of the LENGTH parameter is 65 535 (X'FFFF').

**BLK=RPL**

Specifies that an RPL will be generated.

**ACB=addr**

Specifies the address of the ACB that identifies the data set to which access is requested.

**LEVEL**

Specifies the data level or register to be used for input/output (I/O) as follows:

**Dx** A data level is needed for the VGETC macro to get a record.

**Note:** The data level may be shared by more than one RPL.

**Rx** You can enter the equate value for the level in a register to help in loop processing.

**AREA**

Specifies the address of the data record that was retrieved for I/O requests that were processed with the OPTCD parameter set to LOC. For I/O requests that were processed with the OPTCD parameter set to MVE, VSAM database support moves the record into this work area.

**AREALEN**

Specifies the length, in bytes, of the work area whose address is specified by the AREA parameter. For I/O requests that were processed with the OPTCD parameter set to LOC, the area must be 4 bytes long to contain the address of a data record in the I/O buffer. For I/O requests that were processed with the OPTCD parameter set to MVE, the length must be at least the size of the record on disk; otherwise, an error will be returned.

**ARG**

Specifies the address of a field that contains the search argument for direct retrieval or positioning. For keyed access (that is, the request was processed with the OPTCD parameter set to KEY) the search argument is a full or generic key. If you specify a generic key (that is, the request was processed with the OPTCD parameter set to GEN) you must also specify in the KEYLEN parameter how many of the bytes of the full key you are using for the generic key. For addressed access (that is, the request was processed with the OPTCD parameter set to ADR) the search argument is an RBA.

**KEYLEN**

Specifies the length, in bytes, of the generic key (OPTCD=GEN) that you are using for a search argument. (The search argument is in the field addressed by the ARG parameter.) This parameter is required with a search argument that is a generic key. The value of the KEYLEN parameter must be from 1 to 255. For full-key searches, VSAM database support gets the key length from the catalog definition of the data set when you open that data set.

**RECLLEN**

Specifies the address of a field to contain the length of the record that was retrieved from disk. The field is 2 bytes in length.

**OPTCD**

Specifies the parameters that govern the request.

**ADR**

Specifies addressed access to a key-sequenced or entry-sequenced data set. RBAs are used as search arguments and sequential access is by entry sequence.

**KEY**

Specifies keyed access to a key-sequenced data set. Keys are used as search arguments and sequential access is by key or RBA.

**DIR**

Specifies direct access to a key-sequenced or entry-sequenced data set.

**SEQ**

Specifies sequential access to a key-sequenced data set.

**KEQ**

Specifies that the key search criteria is met when the key of a record is equal to the key that was provided as a search argument (full or generic).

**KGE**

Specifies that the key search criteria is met when the key of a record is greater than or equal to the key that was provided as a search argument (full or generic).

**FKS**

Specifies that a full key is provided as a search argument.

**GEN**

Specifies that a generic key is provided as a search argument.

**LOC**

Specifies that VSAM database support leaves the data record in its I/O buffer for processing.

**MVE**

Specifies that VSAM database support moves the data record into the user-specified work space as defined in the AREA and AREALEN parameters.

**SYN**

Specifies that processing is synchronous (that is, the FINWC macro is used to get a record).

**FWD**

Specifies that sequential processing is in a forward direction.

## Entry Requirements

- Register 9 (R9) must point to an entry control block (ECB) that contains a valid VSAM context block anchored in field CE2VSAM.
- Registers 11 and 12 (R11 and R12) must contain the standard TPF ECB page index values for ECB pages 2 and 3, respectively.

## Return Conditions

- A condition code of 0 indicates that the ACB or RPL was allocated. Register 14 (R14) points to the ACB or RPL.
- A nonzero condition code indicates that the ACB or RPL was not allocated.
- R15 contains the error code.

## Programming Considerations

- This macro can be processed on any I-stream.
- This macro can be called only from E-type programs.
- A maximum of 16 ACBs can be generated automatically by a single ECB.
- A maximum of 32 RPLs can be generated in automatic storage by a single ECB.
- Parameters for VSAM requests are resolved during VGENC macro processing. This means that register parameters only need to be set to their values across the VGENC macro call; they do not need to maintain values for following calls, such as a VGETC macro call. Accordingly, the value of a register parameter at the time of a subsequent VGETC macro call is not relevant.
- The contents of R10, R14, and R15 are destroyed; the contents of all other registers are preserved.

## Examples

- The following example generates an ACB from VSAM automatic storage. The access is keyed and processing is direct. The data definition (DD) name is located at label TRANSLOG.

```

 .
 .
 .
 VGENC BLK=ACB, Generate an ACB
 DDNAME=DDNAME, ..DDname
 MACRF=(KEY,DIR,IN) ..Options

 BNZ ERROR Process Errors
 ST R14,EBW000 Save ACB pointer in ECB
 .
 .
 .
 DDNAME DC CL8'TRANSLOG' Data Definition Name

```

## VGENC

- The following example generates an ACB in location EBW000. The access is keyed and processing is sequential. R3 points to the DD name. The length is specified as ACBSIZE, which is an equate for the size of the ACB in the VSACB DSECT.

```

.
.
.
LA R3,DDNAME Point to DDname string
VGENC BLK=ACB, Generate an ACB
 AM=VSAM, ..Access method is VSAM
 DDNAME=(R3), ..DDname
 MACRF=(KEY,SEQ,IN), ..Options
 WAREA=EBW000, ..Storage area for ACB
 LENGTH=ACBSIZE ..Size of ACB storage area
BNZ ERROR Process Errors
ST R14,EBW000 Save ACB pointer in ECB
.
.
.
DDNAME DC CL8'TRANSLOG' Data Definition Name

```

- In the following example, VSAM database support allocates space for the RPL. (This is denoted by omitting the WAREA parameter.) The ACB to which it refers starts at location EBX000. The data level to be used for I/O is level 4 and the key is at location EBW040. Access is by key and processing is direct. Locate mode is used to retrieve the record. The KGE parameter indicates that an inexact key is given; the KGE parameter directs VSAM database support to return the first key that is equal to or greater than the key in location EBW040. On output, register 2 will be loaded with the address of the data record that was retrieved as well as the length of the record that is placed at location EBW038–039.

```

.
.
.
MVC EBW040(8),KEYVALUE Setup search key argument
LA R2,EBW034 4 byte area - holds data pointer
LA R5,D4 Data Level for RPL operations
VGENC BLK=RPL, Generate an RPL
 LEVEL=(R5), ..Data Level is D4
 ACB=EBX000, ..ACB location in ECB
 AM=VSAM, ..Access method is VSAM
 AREA=(R2), ..Area to hold data pointer
 AREALEN=4, ..data pointer is 4 bytes
 RECLEN=EBW038, ..Put 2 byte data length here
 ARG=EBW040, ..Pointer to search argument
 OPTCD=(KEY,DIR,SYN,LOC,KGE) ..Key search,direct access,
 ..synchronous,return buffer,
 ..search for key >= argument
BNZ ERROR Process Errors
ST R14,EBW000 Save ACB pointer in ECB
.
.
.
DDNAME DC CL8'TRANSLOG' Data Definition Name
KEYVALUE DC CL8'000000001' Search Key value

```

- In the following example, VSAM database support allocates space for the RPL. (This is denoted by omitting the WAREA parameter.) Register 6 points to the ACB to which it refers. The data level to be used for I/O is level 3 and the key is at displacement 15 from register 5. Access is by key and processing is direct. Locate mode is used to retrieve the record. The KEQ parameter indicates that an inexact key is given; the KEQ parameter directs VSAM database support to return the record whose key matches the input key exactly. On output, VSAM database support places the address of the record that was retrieved at

displacement 16 from register 4; the length of that record is placed at the halfword at displacement 20 from register 4. R4, R5, and R6 only need to be set to their values during the VGENC macro call.

```

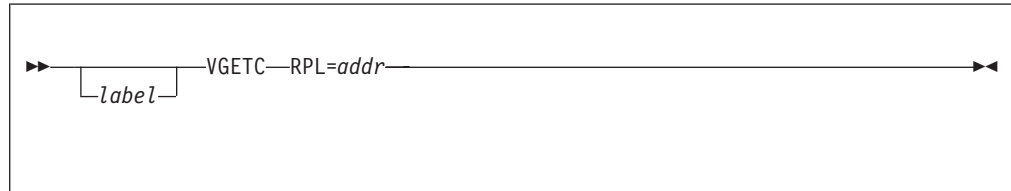
.
.
.
VGENC BLK=RPL, Generate an RPL
 AM=VSAM, ..Access method is VSAM
 ACB=(R6), ..ACB pointer is in R6
 LEVEL=D3, ..Data Level is D3
 AREA=16(R4), ..Area to hold data pointer
 AREALEN=4, ..data pointer is 4 bytes
 RECLEN=20(R4), ..Put 2 byte data length here
 ARG=15(R5), ..Pointer to search argument
 OPTCD=(KEY,DIR,LOC,KEQ) ..Key search, direct access,
 ..return data buffer,
 ..search for key = argument

BNZ ERROR Process Errors
LR R7,R14 Save RPL pointer in R7
.
.
.
```

## VGETC—Get a Record

Use this general macro to get a virtual storage access method (VSAM) database support record.

### Format



#### *label*

An optional label can be used with this macro.

#### *addr*

This parameter specifies the address of the request parameter list (RPL) for this VGETC macro request. You can specify the address in register notation (by using a register enclosed in parentheses) or by using an expression that generates an addressable data area.

### Entry Requirements

- Register 9 (R9) must point to an entry control block (ECB) that contains a valid VSAM context block anchored in field CE2VSAM.
- Registers 11 and 12 (R11 and R12) must contain the standard TPF ECB page index values for ECB pages 2 and 3, respectively.

### Return Conditions

- A condition code of 0 indicates that the VSAM record was retrieved successfully.
- A nonzero condition code indicates that the VSAM record was not retrieved successfully.
- Register 15 (R15) contains the error code.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called only by E-type programs.
- The contents of R10, R14, and R15 are destroyed; the contents of all other registers are preserved.

### Examples

In the following example, a VGETC macro is used to retrieve all records by key sequentially. Retrieval is in a forward direction and processing is synchronous. Because the ARG parameter was not specified, VSAM database support starts at the beginning of the file. (Specifying 0 for the ARG and KGE parameters would also have the same effect.)

```

 .
 .
 .
 VGENC BLK=ACB, Generate an ACB
 DDNAME=DDNAME, ..DDname
 MACRF=(KEY,SEQ,IN) ..Access is Keyed-Sequential

```

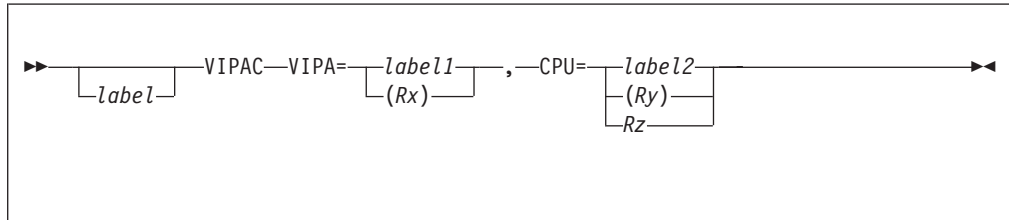


|        |       |                                                                                                         |                                                                                                                                                                                                                                                                 |
|--------|-------|---------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | BNZ   | ERROR                                                                                                   | Process Errors                                                                                                                                                                                                                                                  |
|        | L     | R6,R14                                                                                                  | Save ACB pointer in R7                                                                                                                                                                                                                                          |
|        | VGENC | BLK=RPL,<br>AM=VSAM,<br>ACB=(R6),<br>LEVEL=D3,<br>AREA=EBW000,<br>AREALEN=4,<br>OPTCD=(KEY,SEQ,SYN,LOC) | Generate an RPL<br>..Access method is VSAM<br>..ACB pointer is in R6<br>..Data Level is D3<br>..Area to hold data pointer<br>..data pointer is 4 bytes<br>..Key search, sequential access,<br>..return data buffer, synchronous<br>..start scan at first record |
|        | BNZ   | ERROR                                                                                                   | Process Errors                                                                                                                                                                                                                                                  |
|        | LR    | R7,R14                                                                                                  | Save RPL pointer in R7                                                                                                                                                                                                                                          |
| LOOP   | DS    | 0H                                                                                                      |                                                                                                                                                                                                                                                                 |
|        | VGETC | RPL=(R7)                                                                                                | Get next record in sequence                                                                                                                                                                                                                                     |
|        | BNZ   | CHKE0D                                                                                                  | Check for End of Data                                                                                                                                                                                                                                           |
|        | L     | R4,EBW000                                                                                               | Point to data record                                                                                                                                                                                                                                            |
|        | ...   |                                                                                                         |                                                                                                                                                                                                                                                                 |
|        | B     | LOOP                                                                                                    | Continue scan                                                                                                                                                                                                                                                   |
|        | .     |                                                                                                         |                                                                                                                                                                                                                                                                 |
|        | .     |                                                                                                         |                                                                                                                                                                                                                                                                 |
|        | .     |                                                                                                         |                                                                                                                                                                                                                                                                 |
| DDNAME | DC    | CL8'TRANSLOG'                                                                                           | Data Definition Name                                                                                                                                                                                                                                            |

## VIPAC—Move a VIPA to Another Processor

This general macro moves a virtual IP address (VIPA) to another processor in the same loosely coupled complex.

### Format



*label*

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

**VIPA**

specifies the VIPA that is being moved.

*label1*

is the label that contains the VIPA (previously converted to hexadecimal) that is being moved.

*Rx* is the register in the range R1–R7 that contains the pointer to the VIPA (previously converted to hexadecimal) that is being moved.

**CPU**

specifies the TPF processor to which you want to move the VIPA.

*label2*

is the label that contains the processor ID of the TPF processor to which you are moving the VIPA.

*Ry* is the register in the range R1–R7 that contains the pointer to the processor ID of the TPF processor to which you are moving the VIPA.

*Rz* is the register in the range R1–R7 that contains the processor ID of the TPF processor to which you are moving the VIPA.

### Entry Requirements

- Register 9 (R9) must contain the address of the entry control block (ECB) being processed.
- The system must be in 1052 state or above.

### Return Conditions

- Control is returned to the next sequential instruction (NSI).
- The contents of R14 and R15 are unknown. The contents of R1–R7 are preserved across this macro call.
- R0 will contain one of the following return codes:
 

|          |                              |
|----------|------------------------------|
| <b>0</b> | Move started.                |
| <b>1</b> | Incorrect VIPA.              |
| <b>2</b> | VIPA not defined as movable. |
| <b>3</b> | Incorrect CPU.               |

- 4 CPU not active.
  - 5 System not in 1052 state or above.
  - 6 OSA-Express support is not defined.
  - 7 VIPA not defined on the specified processor.
  - 8 VIPA already moving.
  - 9 CPU already owns the VIPA.
  - 10 Internal system error.
- On return, the protect key is set to working storage.

## Programming Considerations

- This macro can be run on any I-stream.
- To move a VIPA, you must define it as movable to the processor where you want it moved. See *TPF Migration Guide: Program Update Tapes*, *TPF Operations*, and *TPF Transmission Control Protocol/Internet Protocol* for more information about moving VIPAs.

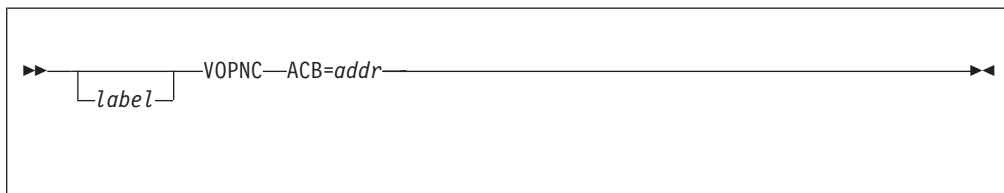
## Examples

- The following example attempts to move the VIPA pointed to in R2 to the processor ID contained in R3.  
VIPAC VIPA=(R2),CPU=R3
- The following example attempts to move the VIPA pointed to in R2 to the processor ID pointed to in R3.  
VIPAC VIPA=(R2),CPU=(R3)
- The following example attempts to move the VIPA stored at label EBW012 to the processor ID stored at label EBW011.  
VIPAC VIPA=EBW012,CPU=EBW011

## VOPNC—Open a Data Set

Use this general macro to open a virtual storage access method (VSAM) data set.

### Format



#### *label*

An optional label can be used with this macro.

#### *addr*

This parameter specifies the address of the access method control block (ACB) for the data set that is being opened. You can specify the address in register notation (by using a register enclosed in parentheses) or by using an expression that generates an addressable data area.

### Entry Requirements

- Register 9 (R9) must point to an entry control block (ECB) that contains a valid VSAM context block anchored in field CE2VSAM.
- Registers 11 and 12 (R11 and R12) must contain the standard TPF ECB page index values for ECB pages 2 and 3, respectively.

### Return Conditions

- A condition code of 0 indicates that the data set was opened successfully.
- A nonzero condition code indicates that the data set was not opened successfully.
- Register 15 (R15) contains the error code.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called only by E-type programs.
- The contents of R10, R14, and R15 are destroyed; the contents of all other registers are preserved.

### Examples

In the following example, two data sets are opened.

```

 .
 .
 .
 VGENC BLK=ACB, Generate an ACB
 DDNAME=TPFPNID, ..DDname
 MACRF=(KEY,DIR,IN) ..Options

 BNZ ERROR Process Errors
 LR R1,R14 Save ACB (1) pointer in R1

 VGENC BLK=ACB, Generate an ACB
 DDNAME=TPFPNIG, ..DDname
 MACRF=(KEY,DIR,IN) ..Options

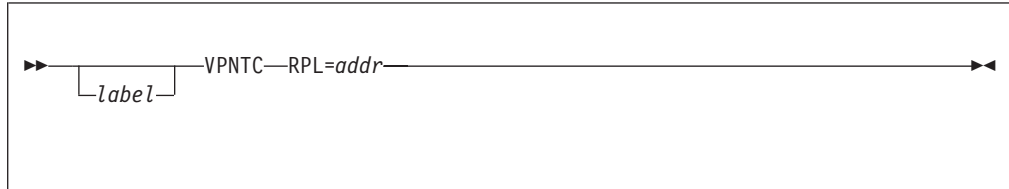
```

|         |       |               |  |                            |
|---------|-------|---------------|--|----------------------------|
|         | BNZ   | ERROR         |  | Process Errors             |
|         | LR    | R2,R14        |  | Save ACB (2) pointer in R2 |
|         | VOPNC | ACB=(R1)      |  | Open PNIDFILE              |
|         | BNZ   | ERROR         |  | ..Open error               |
|         | VOPNC | ACB=(R2)      |  | Open PNIGFILE              |
|         | BNZ   | ERROR         |  | ..Open error               |
|         | .     |               |  |                            |
|         | .     |               |  |                            |
| TPFPNID | DC    | CL8'PNIDFILE' |  | Data Definition Name (1)   |
| TPFPNIG | DC    | CL8'PNIGFILE' |  | Data Definition Name (2)   |

## VPNTC–Point for Access

Use this general macro to position access for subsequent sequential or keyed retrieval. This macro is useful in partitioning scans of large databases; for example, an entry control block (ECB) can scan a range of keys starting at a discrete point in the file.

### Format



#### *label*

An optional label can be used with this macro.

#### *addr*

This parameter specifies the address of the request parameter list (RPL) for this VPNTC macro request. You can specify the address in register notation (by using a register enclosed in parentheses) or by using an expression that generates an addressable data area.

### Entry Requirements

- Register 9 (R9) must point to an entry control block (ECB) that contains a valid VSAM context block anchored in field CE2VSAM.
- Registers 11 and 12 (R11 and R12) must contain the standard TPF ECB page index values for ECB pages 2 and 3, respectively.

### Return Conditions

- A condition code of 0 indicates that the VPNTC macro was processed successfully.
- A nonzero condition code indicates that the VPNTC macro was not completed successfully.
- Register 15 (R15) contains the error code. An end-of-file error is returned if the key value is too high.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called only by E-type programs.
- The contents of R10, R14, and R15 are destroyed; the contents of all other registers are preserved.

### Examples

In the following example, the VPNTC macro positions at the first record that has a key greater than or equal to 5555. The program then scans the database from that point.

**Note:** There is no need to load the key value again each time the VGETC macro is called; VSAM database support maintains context for *next* processing.

```

.
.
.
VGENC BLK=ACB, Generate an ACB
 DDNAME=DDNAME, ..DDname
 MACRF=(KEY,SEQ,IN) ..Options

BNZ ERROR Process Errors
LR R6,R14 Save ACB pointer in R6

MVC EBW000(4),KEYVALUE Setup search key argument
VGENC BLK=RPL, Generate an RPL
 AM=VSAM, ..Access method is VSAM
 ACB=(R6), ..ACB pointer is in R6
 LEVEL=D3, ..Data Level is D3
 ARG=EBW000, ..Search argument starts here
 OPTCD=(KEY,SEQ,KGE,FKS) ..Key search, sequential access,
 ..search for key >= arguments,
 ..do full key search

BNZ ERROR Process Errors
LR R7,R14 Save RPL pointer in R7

VOPNC ACB=(R6) Open file
BNZ ERROR ..Open error

VPNTC RPL=(R7) Position for start of SCAN
BNZ ERROR ..locate error

SCAN DS 0H
 VGETC RPL=(R7) Retrieve next record
 BNZ ENDSCAN ..Assume End of Data

 ... otherwise process record

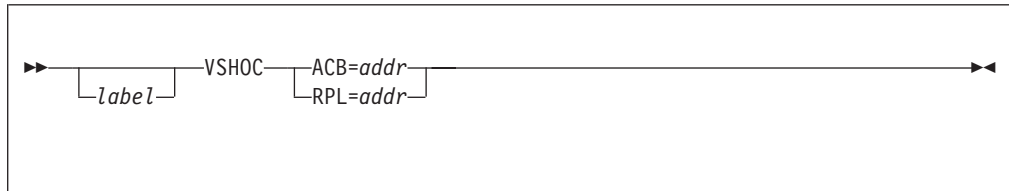
 B SCAN Continue scan
 .
 .
 .
DDNAME DC CL8'TRANSLOG' Data Definition Name
KEYVALUE DC CL4'5555' Search Key value

```

## VSHOC–Display a Control Block

Use this general macro to display the address of the first access method control block (ACB) or request parameter list (RPL) for an entry control block (ECB). The ACB chain field (ACBCHAIN) points to the next ACB in the chain; the RPL chain field (RPLCHAIN) points to the next RPL in the chain. All other ACBs must be accessed by using the ACBCHAIN field; all other RPLs must be accessed by using the RPLCHAIN field.

### Format



*label*

An optional label can be used with this macro.

**ACB=addr**

This parameter specifies a 4-byte area in which to receive the address of the ACB. Use the ACBCHAIN field to access subsequent ACBs.

**RPL=addr**

This parameter specifies a 4-byte area in which to receive the address of the first RPL. Use the RPLCHAIN field to access subsequent RPLs.

### Entry Requirements

- Register 9 (R9) must point to an entry control block (ECB) that contains a valid VSAM context block anchored in field CE2VSAM.
- Registers 11 and 12 (R11 and R12) must contain the standard TPF ECB page index values for ECB pages 2 and 3, respectively.

### Return Conditions

- A condition code of 0 indicates that the VSHOC macro processed successfully.
- A nonzero condition code indicates that the VSHOC macro did not find the ACB or RPL.
- Register 15 (R15) contains the error code.

### Programming Considerations

- This macro can be run on any I-stream.
- This macro can be called only by E-type programs.
- Each RPL maintains a pointer to the ACB to which it refers in field RPLACB.
- The contents of R10, R14, and R15 are destroyed; the contents of all other registers are preserved.

### Examples

The following example displays the address of the first ACB and RPL for an ECB.

•  
•  
•

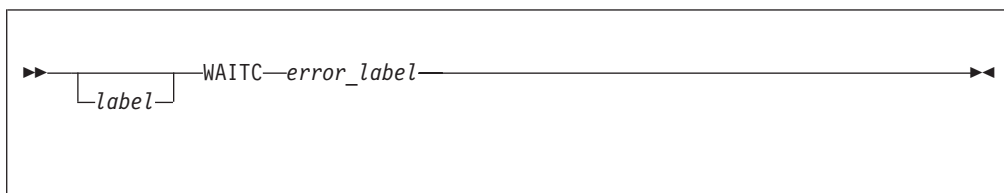


|        |       |                                                  |                                                                                          |
|--------|-------|--------------------------------------------------|------------------------------------------------------------------------------------------|
|        | VGENC | BLK=ACB,<br>DDNAME=DDNAME,<br>MACRF=(KEY,DIR,IN) | Generate an ACB<br>..DDname<br>..Options                                                 |
|        | BNZ   | ERROR                                            | Process Errors                                                                           |
|        | LR    | R6,R14                                           | Save ACB pointer in R6                                                                   |
|        | VOPNC | ACB=(R6)                                         | Open file                                                                                |
|        | BNZ   | ERROR                                            | Process Errors                                                                           |
|        | VGENC | BLK=RPL,<br>AM=VSAM,<br>ACB=(R6),<br>LEVEL=D1    | Generate an RPL<br>..Access method is VSAM<br>..ACB pointer in R6<br>..Use data level D1 |
|        | VSHOC | ACB=0(R6)                                        | Show ACB block chain                                                                     |
|        | BNZ   | ERROR                                            | Process Errors                                                                           |
|        | VSHOC | RPL=0(R7)                                        | Show RPL block chain                                                                     |
|        | BNZ   | ERROR                                            | Process Errors                                                                           |
|        | .     |                                                  |                                                                                          |
|        | .     |                                                  |                                                                                          |
| DDNAME | DC    | CL8'TRANSLOG'                                    | Data Definition Name                                                                     |

## WAITC-Suspend Processing for ECB I/O Completion

This general macro suspends further processing of an entry until all pending input and output requests in process for the entry have been serviced. If a hardware error or unusual condition occurs, a transfer to an ECB program error routine specified by the macro parameters results on return from the WAITC service.

## Format



*label*

A symbolic name may be assigned to the macro statement.

*error label*

The label of an error routine in the current program segment.

## Entry Requirements

R9 must contain the address of the ECB being processed.

## Return Conditions

- Control is returned to the next sequential instruction for successfully completed I/O operations.
- If a hardware error or an unusual condition occurred, control is given to the error routine specified by *error label*.

**Note:** For I/O hardware errors, the system has taken a storage dump and informed CRAS. Detailed information has been stored in the ECB.

- The contents of R14 and R15 are unknown. The contents of all other registers are preserved across this macro call.

## Programming Considerations

- A single WAITC can be used to handle multiple I/O requests.
- If no I/O is pending, there is no loss of control by the entry.
- Pending I/O is tracked in the ECB CE1IOC field.
- All input and output operations that post status back to the entry **require** running WAITC.
- On output operations where the data is detached from the ECB at macro service time, the program cannot determine when I/O operations are completed and no I/O tracking is maintained; therefore, a WAITC is not needed.
- If I/O is pending for this entry, following the execution of this macro, control may be transferred to another entry (ECB). This may result in a transfer to the same program, or another program, on behalf of another entry. This is why this programs must be reentrant.
- Running this macro resets the 500-millisecond program time-out if I/O operations are pending for this ECB.

- Successful completion of a WAITC on 3480 buffered tape writes indicates the data has been written into the control unit buffer but not physically to the tape.
- This macro can be executed on any I-stream.

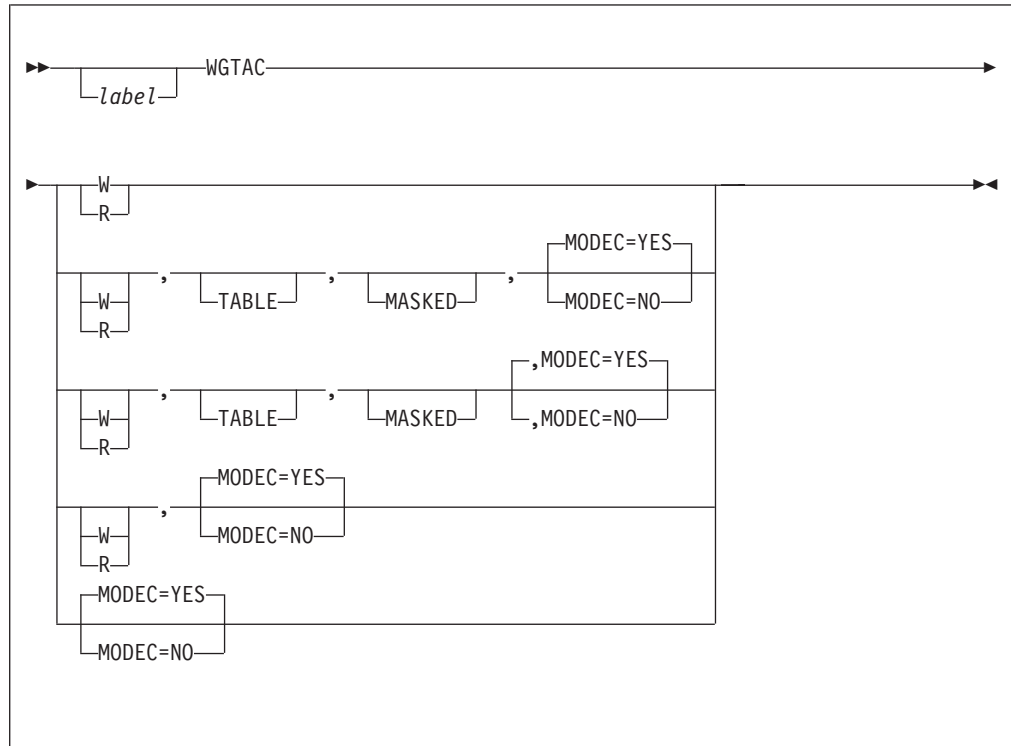
## Examples

None.

## WGTAC—Locate Terminal Entry

This general macro has two functions. The primary function is to locate the entry for a specific line number, interchange address, terminal address (LNIATA) in the terminal address table (WGTA). It also provides the user with the address and the equates for a table describing the (WGTA parameter table).

### Format



#### *label*

A symbolic name can be assigned to the macro statement.

#### **access**

The valid options for ACCESS are either 'R' or 'W'

- R** If the parameter is omitted the default value is 'R'. The R option requests the CINFC macro to be issued with the fast option. (This does not change the storage protect key.)
- W** When the W option is used, the CINFC macro is issued with the write option. (This causes a storage protect key of zero to be placed in the current PSW.) This is the normal usage of the macro. It is issued whenever the user wants to find the address of a specific LNIATA entry in the terminal address subtable.

#### **TABLE**

The omission of TABLE is the normal case. This parameter is coded 'TABLE' when a user wants to find the address in the control program of a table describing the WGTA table.

#### **MASKED**

MASKED is coded 'MASKED' when it is used. This parameter can be used only by C-type segments and is rejected if it is used by E-type (ECB) segments. If

MASKED is coded, the ACCESS and TABLE parameters are ignored. This parameter provides an alternate entry point to the WGTAC service routine for C-type segments.

Segments that use the masked parameter must be running with a storage key that allows alteration of protected storage (key of 0 or F). Use of this parameter allows the interrupt processing routines to run the WGTAC macro.

**MODEC**

Correct values are:

**YES**

WGTAC macro generation issues a MODEC macro to put the user into 31-bit addressing mode. This is necessary to access the WGTA if it resides above the 16-megabyte boundary. YES is the default when MODEC is omitted.

**NO**

WGTAC macro generation does not issue a MODEC. The user is in control of setting the mode.

WGTAC will return in 31-bit mode regardless of the setting of the MODEC parameter if it is invoked in 31-bit mode. If it is invoked in 24-bit mode, it will return in 31-bit mode, unless MODEC is NO, the TABLE parameter is specified and the MASKED parameter is not specified when it will return in 24-bit mode.

**Entry Requirements**

- R9 must contain the ECB address.
- When the TABLE and MASKED parameters are not coded, R1 must contain the LNIATA and CPUID and R2 must be available to this macro routine.
- When only the TABLE parameter is coded, no entry requirements are necessary.
- When MASKED is coded, the ACCESS and TABLE parameters are ignored; R1 must contain the LNIATA and CPUID.

**Return Conditions**

- Control is returned to the next sequential instruction.
- The condition code is unknown.
- When the ACCESS parameter is coded, R1 contains the address of the LNIATA entry in the WGTA table or zeros if the LNIATA was not found. The contents of R2, R14, and R15 are unknown and all other registers are unchanged.
- When the TABLE parameter is coded, R14 contains the address of the table describing the WGTA table in the control program. R15 is unknown and the other registers are unchanged.
- When the MASKED parameter is coded, R1 contains the address of the LNIATA entry in the WGTA table or zeros if the LNIATA was not found. The content of R14 is unknown and all other registers are unchanged.
- When the ACCESS parameter is coded W, a storage protect key of zero is placed in the current PSW.

**Programming Considerations**

- This macro can be executed on any I-stream.
- This macro is coded to be serially reusable only. It can be used by the interrupt processing routines only when the masked parameter is specified.

## WGTAC

- The CINFC macro is used to obtain the address of the WGTA parameter table with all uses of this macro.
- The table parameter should only be used by restart routines when the WGTA table is being constructed.
- The masked parameter should be used only by the interrupt processing routines.
- The MODEC=YES option is ignored in an ISO-C segment (coded with BEGIN TPFISOC=YES).

## Examples

- All defaults WGTAC call

```
LOC WGTAC
```

This call requests read access to the WGTA table. The program issuing the macro is not using the CP-only masked entry point. The macro sets the 31-bit addressing mode switch upon entry.

- Write access to the WGTA table

```
LOCW WGTAC W,MODEC=NO
```

This call requests write access to the WGTA table but doesn't set the 31-bit addressing mode switch. Either the switch has already been set earlier or 24-bit addressing is required by the calling program.

- Read access to the WGTA table

```
TABL WGTAC R, TABLE
```

Read access to the WGTA is requested by this call. A pointer to a table of addresses and constants describing the WGTA area is returned in R14.

- Write access to the WGTA table

```
TABLW WGTAC W, TABLE
```

Similar to the previous example write access to the WGTA is requested by this call. A pointer to a table of addresses and constants describing the WGTA area is returned in R14.

- Masked access

```
MASK WGTAC ,,,MASKED
```

This requests the alternate CP-only entry point. The parameters between the commas default (Read access, no parameter table, addressing mode untouched). If required by the control program, they can be set independently.

## WTOPC—Edit and Send System Message

This general macro edits and sends a message to the system operator or any other designated terminal. It constructs the message block and provides facilities for converting binary values into EBCDIC, decimal, or hexadecimal and for editing the message.

Each time WTOPC is called a single line or a block of text is associated with it. In the discussion that follows we assume each WTOPC is for a single line and use the term *message lines* to refer to lines or blocks.

The WTOPC macro functionally enhances the ROUTC, SENDC, or CSMP interface for system messages. The format of the message produced by the WTOPC macro is in the standard TPF system format:

*xxxxnnnnl hh.mm.ss message text*

Where:

*xxxx*

This is a 4-character message prefix denoting the origin of the message; (for example, VFAC, NETW). The prefix identifies the name of the program issuing the message (unsolicited) or the secondary action code of the operator command the message is in response to. Sometimes the package name is used for the prefix (for example MPIF for messages issued from the MPIF program package).

*nnnn*

This is a 4-digit number assigned to uniquely identify the message from others issued by the same package.

*l* The following are the severity indicators:

- I** Information
- E** Error
- W** Warning
- A** Action required
- T** Termination of function.

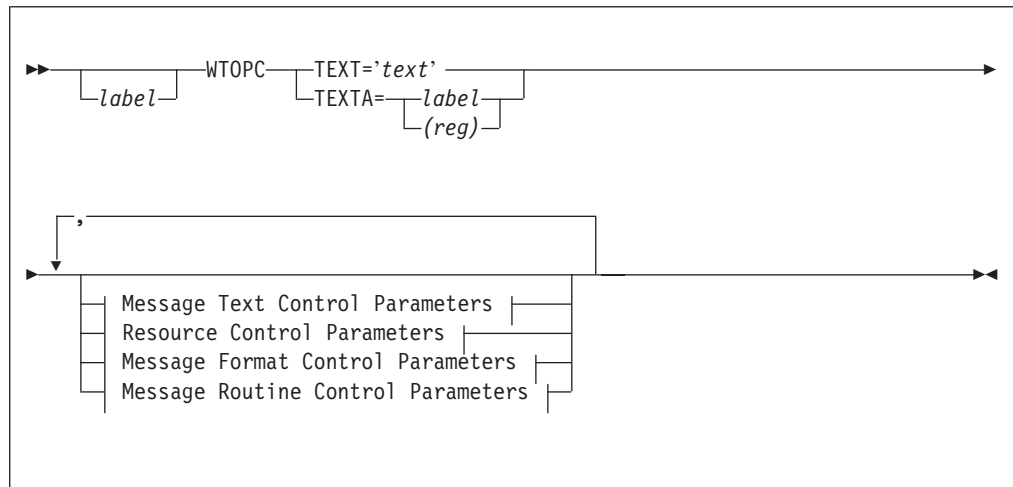
*hh.mm.ss*

This is the time stamp of when the message was issued

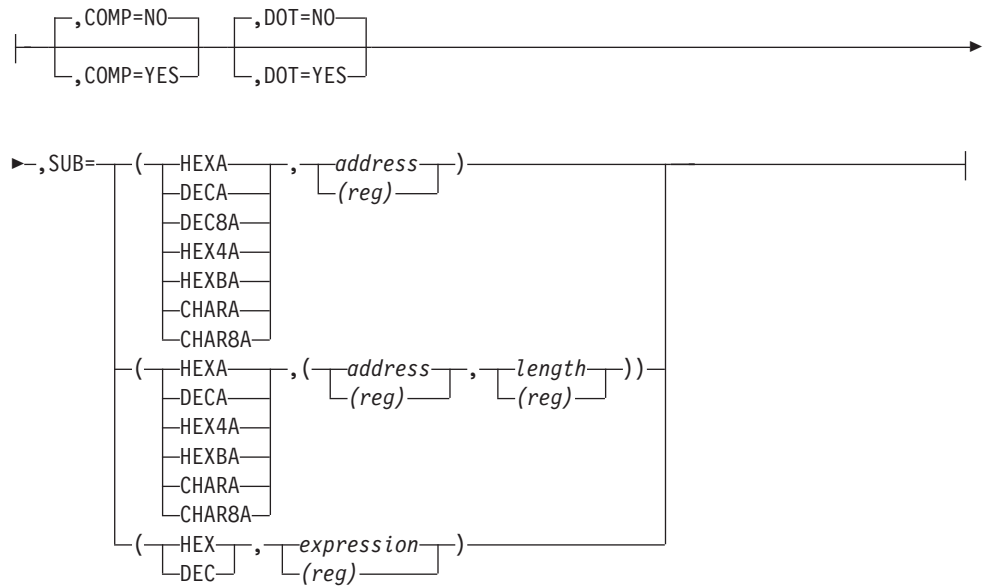
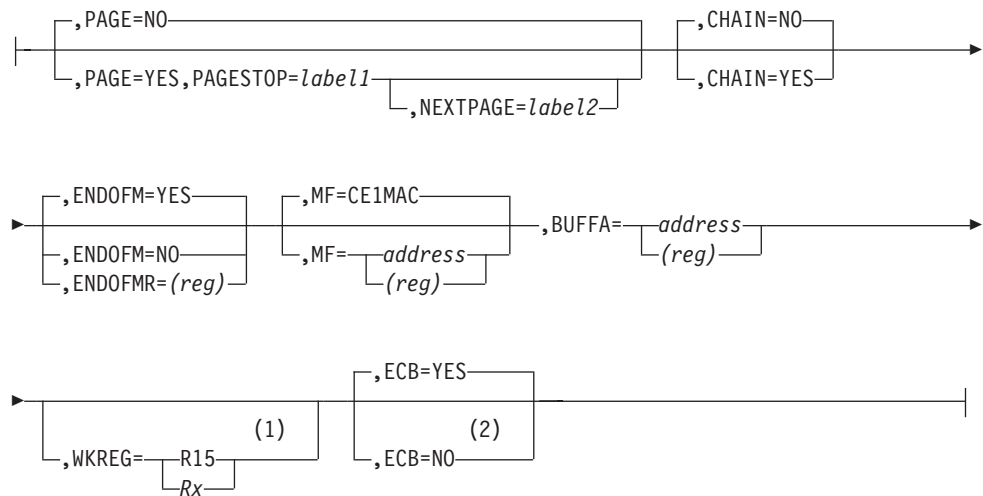
*message text*

This is the text of the message to be sent

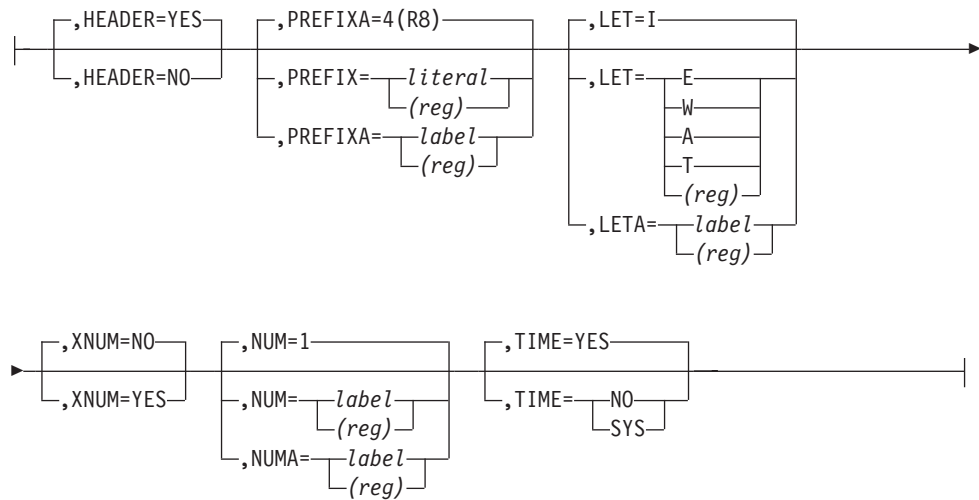
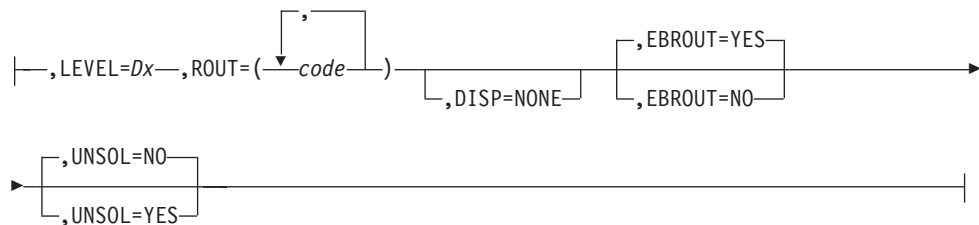
## WTOPC Format





**Message Text Control Parameters:****Resource Control Parameters:****Notes:**

- 1 Allowed on CP invocations only and is required by them.
- 2 Allowed on CP invocations only.

**Message Format Control Parameters:****Message Routing Control Parameters:****label**

A symbolic name can be assigned to the macro statement.

**BUFFA**

This specifies the address of a buffer where the formatted message is to be copied. The message is copied into the indicated buffer, as well as being used as specified in the 'DISP=NONE' operand. The format of the BUFFA operand is:

**address**

The symbolic address of the buffer area.

**(reg)**

A register containing the address of the buffer area.

When the text is copied into the buffer, the length of the message text is inserted into the first byte of the buffer, and the remainder of the text is inserted in subsequent bytes.

**CHAIN**

This specifies whether chaining should take place for the current WTOPC call.

Use the CHAIN operand to link WTOPC calls together to produce an uninterrupted display of message lines in their original order. The format of the CHAIN operand is:

**NO**

Do **Not** chain WTOPC call.

**YES**

This WTOPC call is to be chained to a previous WTOPC call, if any, which also included a CHAIN=YES parameter (during a single ECB).

For example, if you code 3 WTOPC macros with CHAIN=YES on all three, ENDOFM=NO on the first two, and ENDOFM=YES on the last one, the display could be as follows:

```
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the first line of the message
 00.00.00 This is the second line of the message
 00.00.00 This is the third (last) line of the message
```

Although this does not seem any different from the display using just the ENDOFM operand, there are several differences. CHAIN=YES prevents other output messages from interrupting a multiple line display and ensures the output lines appear in the same order that the WTOPC macros were issued. When used with the HEADER and ENDOFM operands, CHAIN=YES can be used to format long displays.

CHAIN=YES is **NOT** allowed when ECB=NO. In other words, CP code executing without an ECB cannot use the CHAIN parameter, but this does not mean that CP code cannot issue a multiple-line message. (Refer to the HEADER parameter to see how this is accomplished.) If CHAIN is coded with ECB=NO, it will be flagged and ignored.

**Note:** CHAIN=YES and UNSOL=YES cannot be specified for the same WTOPC call.

**COMP**

Use the COMP operand when you do not want to display multiple blanks within your message text. The format of the COMP operand is:

**NO**

Do not remove multiple blanks from the message. This is the default if the COMP parameter is not specified.

**YES**

Remove multiple blanks from the message.

For example, if you code:

```
WTOPC TEXT='TOTAL 5',COMP=YES
```

the line is displayed as:

```
TOTAL 5
```

**DISP=NONE**

The DISP parameter specifies the output disposition of the edited line.

When DISP is not coded, the message text is built and sent to the designated terminal.

## WTOPC

When the DISP parameter is coded, the only correct value for it is 'NONE'. 'DISP=NONE' specifies that no output occurs. The message is built, returned to the BUFFA area, and not sent. When 'DISP=NONE' is used, the BUFFA parameter must also be specified.

### DOT

This specifies whether a period is to be placed at the end of the message.

#### NO

Period will not be placed at the end of the message. This is the default if the DOT parameter is not specified.

#### YES

Period will be placed at the end of the message.

For example, if you code:

```
WTOPC TEXT='COMPLETED',DOT=YES
```

the line displayed is:

```
COMPLETED.
```

### EBROUT

This specifies whether WTOPC sends the message to the address contained in EBROUT.

#### YES

Send to the address contained in EBROUT. This is the default if the EBROUT parameter is not specified.

#### NO

Do not use EBROUT as one of the destinations.

**Note:** If EBROUT=NO is coded, the ROUT parameter or the LEVEL parameter must also be coded.

### ECB

This specifies whether CP code is running with or without an ECB. This option is not valid for E-type programs.

#### YES

The running CP code does have an ECB. This is the default when the ECB parameter is not specified.

#### NO

The running CP code does **NOT** have an ECB. CP code calling WTOPC with ECB=NO must be in SVM addressing mode.

CP code normally runs without an ECB. All WTOPC calls from the CP (as well as ECB=NO) must specify UNSOL=YES and XNUM=YES.

ECB creation time is used to identify each chained message, so a WTOPC called by CP code is not able to send chained messages. As a result, the existence or nonexistence of an ECB must be specified by the programmer.

#### Notes:

1. It is still possible for CP code to send multiple-line displays. Refer to the HEADER parameter.
2. If ECB=NO is coded, the ROUT parameter must also be coded.

### ENDOFM

This specifies whether output message lines following the current message line

are prefixed with the standard system header. If used with CHAIN=YES, ENDOFM also specifies whether the current message line is the end of the chain.

Use the ENDOFM operand when you want to suppress prefixing of the message lines following the current message line. The format of the ENDOFM operand is:

#### YES

Do not suppress prefixing of the following message lines. This is the default if the ENDOFM parameter is not specified.

#### NO

Suppress prefixing of message lines following.

For example, if you code ENDOFM=NO, the display will be as follows:

```
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the first message line
 00.00.00 This is the second message line
 00.00.00 This is the third message line
```

Otherwise, if you code ENDOFM=YES (default value) or ENDOFMR=(R6) and R6 contains the value zero, the display will be as follows:

```
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the first message line
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the second message line
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the third message line
```

#### ENDOFMR

This specifies a register the contents of which control whether message lines are prefixed. If the contents of the specified register are:

Nonzero, subsequent lines are not prefixed or  
Zero, subsequent lines are prefixed.

If used with CHAIN=YES, ENDOFMR also specifies whether the current message line is the end of the chain.

Use the ENDOFMR operand when you want to suppress prefixing of the message lines following the current message line. The format of the ENDOFMR operand is:

(reg)

If register contains a value other than zero, prefixing of the message will be suppressed.

For example, if you code ENDOFMR=(R6) and R6 contains a value other than zero, the display will be as follows:

```
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the first message line
 00.00.00 This is the second message line
 00.00.00 This is the third message line
```

If you code ENDOFMR=(R6) and R6 contains the value zero, the display will be as follows:

```
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the first message line
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
```

## WTOPC

```
xxxxnnnnl 00.00.00 This is the second message line
CSMP0097I 00.00.00 CPU-B SS-BSS SSU-HPN IS-01
xxxxnnnnl 00.00.00 This is the third message line
```

### HEADER

This specifies whether or not the output message should contain the header created from the PREFIX(A), NUM(A), LET(A) and TIME operands. Also, for CP code with ECB=NO, specifies whether the output line should be prefixed with the standard system header.

Use the HEADER operand if the header is to be suppressed. The format of the HEADER operand is:

#### YES

Include header in the output message. This is the default if the HEADER parameter is not specified. For CP code with ECB=NO, indicate that this line should be prefixed.

#### NO

Send message without a header. For CP code with ECB=NO, indicate that this line should not be prefixed.

For example, if you code:

```
WTOPC TEXT='FIRST WTOPC MESSAGE',PREFIX =WTOP,NUM=01,LET=I
WTOPC TEXT='SECOND WTOPC MESSAGE',HEADER=NO
```

The following lines are displayed:

```
WTOP0001I 00:00:00 FIRST WTOPC MESSAGE
SECOND WTOPC MESSAGE
```

### LET

This specifies the message severity indicator letter.

#### *literal*

Literal message severity indicator letter:

- I** Information
- E** Error
- W** Warning
- A** Action required
- T** Termination of function

#### *(reg)*

A register containing the indicator letter in bits 24-31.

### LETA

This specifies the address of the message severity indicator letter.

#### *label*

The symbolic address of the severity indicator letter.

#### *(reg)*

A register containing the address of the severity indicator letter.

**Note:** If LET or LETA are not specified and HEADER=YES, the default LET=I is used.

### LEVEL

This specifies the data level of the message block containing the text. This operand is not valid if WTOPC is issued from the control program.

Use the LEVEL operand to pass the text in a 381-byte message block in AM0SG (AM0SG) format. The output text should be in a 381-byte block and the maximum text count must not exceed 335 (UI0TMX).

The message destination is based on the address in the message and routing codes specified by the ROUT parameter, if any. EBROUT is not used. No text manipulation is performed, therefore the following operands will be considered invalid when the LEVEL operand is coded; COMP, DOT, SUB, DISP, BUFFA, MF, TEXT, TEXTA, and ECB. WTOPC returns to the user with the message block detached.

*Dx* A data level, D0–DF, with the message block.

## MF

Use the MF operand to specify a work area other than the one set up by WTOPC.

- For E-type code, the MF parameter may be necessary when substitution parameters override the area provided. Overriding the default macro work area will generate an assembly message. The default work area for E-type programs is CE1MAC.
- For CP code running with an ECB, the MF parameter may be necessary to prevent the use of the macro work area when it is still in use by another macro.
- For CP code running without an ECB, this operand must be used with operands that could cause nonreentrant code to be generated.

The format of the MF option is:

*address*

Generates code to fill in the parameter list at the location in the work area specified by the symbolic address.

*(reg)*

Generates code to fill in the parameter list at the location in the work area specified by the register.

**Note:** The address specified should always be a work area outside the program so that the code is reentrant. For CP code, the work area should be carved out of the stack area. The default work area for C-type programs without an ECB is inline.

## NEXTPAGE=*label*2

This specifies the name of the label in the code that should receive control when an operator issues the ZPAGE command. If no label is specified, control is passed to the next sequential instruction.

**Note:** A value can be specified for this parameter only when PAGE=YES.

## NUM

This specifies the number assigned to the message and is used with the XNUM parameter.

When XNUM=NO is coded, the valid range for NUM is decimal 1–255.

When XNUM=YES is coded, the valid range is decimal 1–9999.

*literal*

Literal message number.

*(reg)*

A register containing a binary number that will be converted from binary to decimal.

When XNUM=NO, the binary number is assumed to be in bits 24–31.

## WTOPC

When XNUM=YES, the binary number is assumed to be in bits 16–31.

### NUMA

This specifies the address of a binary field containing the number assigned to the message. It is used with the XNUM parameter because when XNUM=NO, NUMA must specify a 1-byte field, and when XNUM=YES, NUMA must specify a 2-byte field.

*label*

The symbolic address of the message number.

*(reg)*

A register containing the address of the message number.

**Note:** If neither NUM nor NUMA is specified and HEADER=YES, the default NUM=1 is used.

### PAGE=NO|YES

This specifies whether to display the ENTER ZPAGE TO CONTINUE message and suspend the ECB when the number of lines in the message exceeds the size of the WTOPC output page. In this case, the operator must issue the ZPAGE command in order to continue displaying the message.

**Note:** If PAGE=YES, then CHAIN=YES and ECB=YES must also be specified.

### PAGESTOP=*label*

This specifies the name of the label in the code that should receive control if:

- The operator does not issue the ZPAGE command in 1 minute
- Before issuing the ZPAGE command, the operator issues another command from the same terminal that also requires the operator to issue ZPAGE.

When this label receives control, the program should not issue any more output.

#### Notes:

1. A value must be specified for this parameter when PAGE=YES.
2. Since the ECB is being suspended during processing of a PAGESTOP, the ECB must NOT be holding any resources (gained through functions such as FIWHC, CORHC, and LOCKC). Holding resources could result in a lock out condition.

### PREFIX

This specifies the 4-character message prefix.

*literal*

The literal message prefix.

*(reg)*

A register containing the message prefix.

Use the PREFIX operand to precede the message with four characters denoting its origin.

For example, if you code:

```
WTOPC TEXT='OUTPUT MESSAGE',PREFIX=CVAD,NUM=01,LET=I,...
WTOPC TEXT='OUTPUT MESSAGE 2',PREFIX=(R7),NUM=02,LET=I,...
```

where CVAD is the desired message prefix and R7 contains X'C3E5D1C3' (C'CVJC'), the following lines are displayed:



```
CVAD0001I 00.00.00 OUTPUT MESSAGE 1
CVJC0002I 00.00.00 OUTPUT MESSAGE 2
```

**PREFIXA**

This specifies the address of a 4-character field containing the message prefix.

*label*

The symbolic address of the message prefix.

*(reg)*

A register containing the address of the message prefix.

Use the PREFIXA operand to precede the message with four characters denoting its origin.

For example, if you code the following:

```
WTOPC TEXT='OUTPUT MESSAGE 1',PREFIXA=PREFIX,NUM=01,LET=I,...
WTOPC TEXT='OUTPUT MESSAGE 2',PREFIXA=EBW068,NUM=02,LET=I,...
WTOPC TEXT='OUTPUT MESSAGE 3',PREFIXA=(R6),NUM=03,LET=I,...
```

where:

PREFIX DC CL4'CVLC',

EBW068 contains X'C3E5D8C3' (C'CVQC'), and

R6 contains the address of a 4-byte, character field containing X'C3E5D4C7' (C'CVMG')

the following lines are displayed:

```
CVLC0001I 00.00.00 OUTPUT MESSAGE 1
CVQC0002I 00.00.00 OUTPUT MESSAGE 2
CVMG0003I 00.00.00 OUTPUT MESSAGE 3
```

**Note:** If either PREFIX or PREFIXA are not specified and HEADER=YES, the default will be the program name, PREFIXA=4(R8). This is the TPF system standard for message identifiers.

**ROUT=code**

This specifies which functional support console (FSC) will receive the message. Valid FSC names for this parameter are defined in the RTCEQ macro.

Use the ROUT operand to specify specific functional support consoles to which the message should be sent. You may specify either the mnemonic, as defined in RTCEQ, or the actual functional support console (1–16).

ROUT=(routing\_code<sub>1</sub>,...,routing\_code<sub>n</sub>...)

Routing codes specified on the WTOPC macro are in addition to any functional support console (FSC) bits set upon entry.

**SUB**

This specifies a substitution list describing the conversions to be performed on the message.

The SUB parameter specifies the type of substitution performed on portions of the message indicated by periods. For each set of periods, specify the type of substitution and the value to be substituted or its address. If the message text does not contain periods, no substitution is performed.

You can specify the SUB parameter in one of the following formats:

- Value substitution. Specify this in one of the following ways:
  - SUB=(type,expression)

– SUB=(*type*,(*reg*))

Where:

*type*

Specifies the value HEX or DEC.

*expression*

Specifies the value of an expression to be substituted.

(*reg*)

Specifies a register whose contents are to be substituted.

- Address substitution; implied length. Specify this in one of the following ways:

– SUB=(*type*,*address*)

– SUB=(*type*,(*reg*))

Where:

*type*

Specifies the value HEXA, DECA, DEC8A, HEX4A, HEXBA, CHARA, or CHAR8A.

*address*

Specifies the address where the data to be substituted is located.

(*reg*)

Specifies a register that contains the address of the data to be substituted.

- Address substitution; explicit length. Specify this in one of the following ways:

– SUB=(*type*,(*address*,*length*))

– SUB=(*type*,((*reg1*),*length*))

– SUB=(*type*,(*address*,(*reg2*)))

– SUB=(*type*,((*reg1*),(*reg2*)))

Where:

*type*

Specifies the value HEXA, DECA, DEC8A, HEX4A, HEXBA, CHARA, or CHAR8A.

*address*

Specifies the address where the data to be substituted is located.

(*reg1*)

Specifies a register that contains the address of the data to be substituted.

*length*

Specifies the length of the data to be substituted.

(*reg2*)

Specifies a register that contains the length of the data to be substituted.

The following describes each of the possible substitution pairs. These descriptions are followed by information about length specification and multiple substitution lists.

**HEX**,(*reg*)

Converts the value in the specified register to graphic hexadecimal format

and substitutes it in the message text. If you code fewer than eight consecutive periods in the message text, then leading digits are truncated; leading zeros are not suppressed.

For example, if register 3 contains the value C0031FC8, the following macro statement:

```
WTOPC TEXT='VALUE = ...',SUB=(HEX,(R3))
```

results in the display:

```
VALUE = FC8
```

#### **HEX,expression**

Converts the given expression to graphic hexadecimal format and substitutes it in the message text. The variable expression, may be a symbolic address or a symbol equate; it is evaluated by means of a LOAD ADDRESS (LA) instruction. For example, if your program has a label BUFF1, the following macro statement:

```
WTOPC TEXT='BUFFER IS LOCATED AT',SUB=(HEX,BUFF1)
```

can result in the display:

```
BUFFER IS LOCATED AT 0201AC
```

If you code fewer than eight periods in the message text, leading digits are truncated; leading zeros are not suppressed.

#### **DEC,(reg)**

Converts the value in the specified register into graphic decimal format and substitutes it in the message text. Leading zeros are suppressed. If the number is negative, a leading minus sign is inserted. For example, if register 3 contains the decimal value 10,345, the following macro statement:

```
WTOPC TEXT='REG 3 =',SUB=(DEC,(R3))
```

results in the display:

```
REG 3 = 10345
```

#### **DEC,expression**

Converts the given expression to graphic decimal format and substitutes it in the message text. The variable expression may be a symbolic label in your program or a symbolic equate. For example, assume your program contains the following statement

```
VALUE EQU 2003
```

In this example, the following macro statement:

```
WTOPC TEXT='VALUE IS',SUB=(DEC,VALUE+5)
```

results in the display:

```
VALUE IS 2008
```

#### **HEXA,address**

Converts the fullword at the specified address to graphic hexadecimal format and substitutes it in the message text. If you code fewer than eight periods in the message text, leading digits are truncated; leading zeros are not removed. For example, if you code the following macro statement, the last five hexadecimal digits of the fullword at the label CODE are substituted into the message text.

```
WTOPC TEXT='HEX VALUE IS',SUB=(HEXA,CODE)
```

**HEXA,(reg)**

Converts the fullword at the address indicated in the specified register into graphic hexadecimal format and substitutes it in the message text. For example, if you code the following macro statement, the last six hexadecimal digits of the fullword the address of which is in register 5 are substituted in the message text.

```
WTOPC TEXT='REGISTER 5 -',SUB=(HEXA,(R5))
```

If you code fewer than eight digits, leading digits are truncated; leading zeros are not suppressed.

**DECA,address**

Converts the fullword at the specified address to graphic decimal format. Leading zeros are suppressed; if the number is negative, a minus sign is inserted. For example, if you code the following macro statement, the fullword at the location COUNT is converted to graphic decimal format and substituted in the message text.

```
WTOPC TEXT='COUNT =',SUB=(DECA,COUNT)
```

**DECA,(reg)**

Converts the fullword at the address specified in the indicated register into graphic decimal format and substitutes it in the message text. For example, the following macro statement causes the value in the fullword the address of which is in register 3 to be displayed in graphic decimal format.

```
WTOPC TEXT='SUM =',SUB=(DECA,(3))
```

**DEC8A,address**

Converts the 8-byte field at the address specified to graphic decimal format and substitutes it in the message text. Leading zeros are suppressed; a blank is inserted every 3 digits going from right to left; if the number is negative, a minus sign is inserted. For example, the following macro statements cause the value in the 8-byte field to be displayed in graphic decimal format.

```
BIGNUM DC XL8'00000001FFFFFFFF'
```

```
WTOPC TEXT='SUM =',SUB=(DEC8A,BIGNUM)
```

results in the display:

```
SUM = 8 589 934 591
```

**DEC8A,(reg)**

Converts the 8-byte field at the address specified in the register indicated into graphic decimal format and substitutes it in the message text. Leading zeros are suppressed; a blank is inserted every 3 digits going from right to left; if the number is negative, a minus sign is inserted. For example, the following macro statements cause the value in the 8-byte field (the address of which is in register 3) to be displayed in graphic decimal format.

```
NUMBER DC XL8'0000000000001000'
```

```
WTOPC TEXT='SUM =',SUB=(DEC8A,NUMBER),COMP=YES
```

results in the display:

```
SUM = 4 096
```

**HEX4A,address**

Converts the data at the specified address to graphic hexadecimal format, and inserts a blank character following every 4 bytes ( 8 characters of output). The data to be converted does not have to be on a fullword boundary.

When you code periods in the message text for substitution, you must code sufficient periods to allow for the blanks. Thus, to display 8 bytes of information (16 hexadecimal digits), you must code 17 periods in the message text.

For example, to display 7 bytes of hexadecimal data beginning at location STOR in your program, you can code the following:

```
WTOPC TEXT='STOR:',SUB=(HEX4A,STOR)
```

This results in the following display:

```
STOR: 0A23F115 78ACFE
```

**Note:** 15 periods were coded in the message text, to allow for the blank following the first 4 bytes displayed.

#### HEX4A,(reg)

Converts the data at the address indicated in the specified register to graphic hexadecimal format and inserts a blank character following every 4 bytes displayed (8 characters of output).

When you code the message text for the substitution, you must code sufficient periods to allow for the blank characters to be inserted.

For example, the following macro statement:

```
WTOPC TEXT='.....',SUB=(HEX4A,(R6))
```

results in the display of the first 9 bytes at the address in register 6, in the following format:

```
hhhhhhhh hhhhhhhh hh
```

#### HEXBA,address

Converts the data at the specified address to graphic hexadecimal format and displays the output contiguously; that is, blank characters are not inserted. The data to be converted does not have to be on a fullword boundary.

For example, to display 7 bytes of hexadecimal data beginning at location STOR in your program, you can code the following:

```
WTOPC TEXT='STOR:',SUB=(HEXBA,STOR)
```

This results in the following display:

```
STOR: 0A23F11578ACFE
```

#### HEXBA,(reg)

Converts the data at the address indicated in the specified register to graphic hexadecimal format and displays the output contiguously; that is, blank characters are not inserted.

For example, the following macro statement:

```
WTOPC TEXT='.....',SUB=(HEXBA,(R6))
```

results in the display of the first 9 bytes at the address in register 6, in the following format:

```
hhhhhhhhhhhhhhhhhh
```

#### CHARA,address

Substitutes the character data at the specified address into the message text. For example, the following macro statement:

```
WTOPC TEXT='NAME IS ''.....'',SUB=(CHARA,NAME)
```

## WTOPC

causes the 10 characters at location name to be substituted into the message text.

### **CHARA,(reg)**

Substitutes the character data at the address indicated in the specified register into the message text. For example, consider the following macro statement:

```
WTOPC TEXT='CODE IS',SUB=(CHARA,(R7))
```

In this example, the first 4 characters at the address indicated in register 7 are substituted in the message line.

### **CHAR8A,address**

Substitutes the character data at the specified address into the message text, and inserts a blank character following each 8 characters of output.

### **CHAR8A,(reg)**

Substitutes the character data at the address indicated in the specified register and inserts a blank character following each eight characters of output.

## **Specifying the Length for WTOPC Macro Substitution**

In all the examples shown, the length of the argument being substituted was determined by the number of periods in the message text. The number of periods indicated the size of the output field, and indirectly determined the size of the input data area.

For hexadecimal and decimal substitutions, the input data is truncated on the left. To ensure that a decimal number will never be truncated, you can code 10 periods (11 for negative numbers) in the message text where it will be substituted. For hexadecimal data, code 8 periods to ensure that no characters are truncated when a fullword is substituted.

When you are coding substitution lists with the CHARA, CHAR8A, HEX4A, and HEX8A options, however, you can specify the length of the input data field. To do so, code the SUB operand as follows:

```
SUB=(type,(address,length))
```

Both address and length may be specified using register notation. For example:

```
SUB=(HEX4A,(LOC,(R4)))
```

shows that the characters at location LOC are substituted into the message text; the number of characters is determined by the value contained in register 4, but it cannot be larger than the number of periods coded in the message text.

You can use this method in the special case where only 1 character is to be substituted. Since you must always code at least two periods to indicate that substitution is to be performed, you can code two periods and specify a length of 1, as follows:

```
WTOPC TEXT='INVALID MODE LETTER ..',SUB=(CHARA,(PLIST+24,1))
```

## **Specifying Multiple Substitution Lists**

When you want to make several substitutions in the same line, you must enter a substitution list for each set of periods in the message text. For example,

WTOPC TEXT='VALUES ARE ..... AND .....',SUB=(DEC,(R3),HEXA,LOC)

might generate a line as follows:

VALUES ARE -45 AND FFE3C2.

**Note:** If the default work area is used to build the parameter list, coding an excessive number of substitutions will overrun the work area and cause a warning message to be generated. To calculate how many bytes will be needed for the work area see "Programming Considerations" on page 577.

#### **TEXT='text'**

Use the TEXT operand to specify the exact text of the message on the macro instruction. The message text must appear within single quotation marks, as follows:

WTOPC TEXT='message text'

If you want a single quotation mark to appear within the actual message text, you must code two of them.

Text specified on the WTOPC macro may be edited so that multiple blanks appear as only a single blank, and a period may be placed at the end of the line, for example:

TEXT='IT ISN'T READY'

will result in:

IT ISN'T READY.

#### **TEXTA**

This specifies the address of the message text.

*label*

The symbolic address of the message text.

*(reg)*

A register contains address of the message text.

Use the TEXTA operand when you want to send a message contained in a buffer. You may specify either a symbolic address or use register notation. In either case, the first byte at the address specified must contain the length of the message text, for example:

WTOPC TEXTA=MESSAGE

where MESSAGE must be defined as follows:

```
MESSAGE DC AL1(MSGLEN-MESSAGE-1),AL1(#CAR)
DC C'LINE ONE',AL1(#CAR)
DC C'LINE TWO'
MSGLEN EQU *
```

will produce:

```
LINE ONE
LINE TWO
```

**Note:** There are 2 additional macros that help the generation of message text for WTOPC output. These macros should be used whenever possible.

#### **GENMSG**

This macro can be used to generate message text, number, and letter data, and is handy for defining

message text tables. See “GENMSG—Generate Message Table for WTOPC” on page 242 for more information.

**DCTMSG**

This is a dsect that defines the data required by GENMSG. Tags in this dsect can be used with the TEXTA, NUMA, and LETA parameters.

**TIME**

This parameter specifies whether or not the message should include a time stamp and whether the time should reflect the system time or the subsystem time.

**NO**

Suppresses time stamping of system messages.

**YES**

The subsystem time will be inserted between the message header and the message text. This is the default if the TIME parameter is not specified.

**SYS**

The system time will be inserted between the message header and the message text.

**Notes:**

1. The TIME parameter is suppressed if HEADER=NO.
2. A time stamp is a required parameter on an output message. If TIME=NO is specified, the WTOPC user should provide a time stamp as part of the output text.

**UNSOL=NO|YES**

This specifies whether messages being sent to remote terminals are sent as unsolicited messages (UNSOL=YES) or solicited messages (UNSOL=NO).

Unsolicited messages are handled by the unsolicited message package (UMP). Operators may need to issue the LOGU command to display unsolicited messages.

**Note:** UNSOL=YES and CHAIN=YES cannot be specified for the same WTOPC call.

**WKREG=reg**

This parameter specifies a work register for WTOPC processing. It is only valid for CP programs and is mandatory for them. WTOPC uses R15 for a work register, which could conflict with CP code using R15 for its base register.

If R15 is available, code WKREG=R15.

If R15 is not available, code WKREG=R<sub>x</sub>, where R<sub>x</sub> is an available register.

**XNUM**

This specifies whether the message number will be passed in a 1- or 2-byte field.

**YES**

The message number is greater than 255. The number will then be passed in a 2-byte field.

**NO**

The number will be passed in a 1-byte field.

XNUM=NO is the default if the XNUM parameter is not specified.



This parameter is affected by any previous use of WTOPC or DCTMSG. For example: if DCTMSG is coded with XNUM=YES, the following WTOPC must code XNUM=YES or omit the XNUM parameter. In this case, omitting XNUM will result in XNUM=YES being the default.

## Entry Requirements

- R9 must contain the address of the ECB being processed for E-type programs and CP code with ECB=YES operand coded.
- WTOPC determines the address of the terminal to receive the output message in the following manner:
  - The terminal address is retrieved from the EBROUT ECB field.
  - When the LEVEL parameter is coded, the address is retrieved from the message block on the specified data level. EBROUT is ignored.
  - When EBROUT=NO or ECB=NO is coded, the address is determined from the ROUT parameter.
- General register R0, R14, and R15 may not be used for passing parameters to the WTOPC macro.
- For CP WTOPCs coded with ECB=NO, the parameter list is built inline.
- For E-type programs, the WTOPC macro requires a work area in which to build a parameter list. If the MF operand is omitted then a reserved macro work area will be used. When the LEVEL operand is specified, the remaining bytes of the message block following the text will be used as the work area.

## Return Conditions

- Control is returned to the next sequential instruction unless PAGE=YES is specified. In this case, control is returned to the label specified for the NEXTPAGE or PAGESTOP parameters.
- The contents of R0, R14, and R15 are unknown. For CP code, R15 will be preserved if the WKREG= parameter is used.
- The contents of all other registers are preserved across this macro call.
- When the LEVEL operand is specified, the message block is detached.
- The status of the operation is not known.

## Programming Considerations

- WTOPC depends on CVIQ and can not be used before CVIQ is in the system. Therefore, WTOPC can not be used by IPL, CTIN, or ALDR.
- If CHAIN=YES and ECB=NO are both coded, the CHAIN operand will be ignored.
- CHAIN=YES and UNSOL=YES cannot be coded for the same WTOPC call.
- If EBROUT and ECB=NO are both coded, the EBROUT operand will be ignored.
- This macro can be executed on any I-stream.
- Because of message routing restrictions, a maximum of 100 blocks can be chained. This does not mean that the maximum number of lines is limited to 100. WTOPC takes the text of each send and packs it into 381-byte file-chained blocks. The maximum of 100 blocks refers to the size of the file chain.
- The message text can contain the following number of characters:
  - When UNSOL=NO
    - If a message header is specified, a maximum of 236 characters.
    - If no message header is specified, a maximum of 256 characters.

## WTOPC

- When UNSOL=YES
  - If a message header is specified, a maximum of 162 characters.
  - If no message header is specified, a maximum of 181 characters.
- When message text for the WTOPC contains two or more consecutive periods, it indicates that a substitution is to be performed on that portion of the message. The number of periods you code indicates the number of characters that you want to appear as output. To indicate which values are to replace the periods, code a substitution list using the SUB operand.
- If the ENDOFM or ENDOFMR parameter is specified as NO, another WTOPC message is expected to send the end-of-message character (ENDOFM or ENDOFMR specified as YES). If there is no subsequent WTOPC message, the receiving terminal may hang waiting for the end-of-message indicator.
- When the 'MF' parameter is not specified, WTOPC uses a reserved macro work area to build the parameter list. If the LEVEL parameter is specified, the remaining bytes of the message block following the text will always be used as the work area. The length of the work area used depends upon the number of substitutions in the message text and can be computed by:
  - 12 + 6 if HEADER=YES
  - + 4 if BUFFA was specified
  - + 5 for each substitution that does not specify an explicit length
  - + 6 for each substitution that does specify an explicit length

If the default work area is used and the parameter list is too large to fit in it because of too many substitutions, a warning message will be generated.

- If WTOPC is issued for a Solicited message using EBROUT=NO, there is the possibility of hanging the originating terminal.
- If PAGE=YES is coded for a multiple line message, each line of the message should be sent using a separate WTOPC call.
- PAGE=YES can be coded only when CHAIN=YES and ECB=YES are coded. The NEXTPAGE and PAGESTOP parameters can be coded only when PAGE=YES is coded. If PAGE=YES is coded, the PAGESTOP parameter is required and the NEXTPAGE parameter is optional.
- PAGE=YES may cause the ECB to be suspended. Therefore, the ECB should not hold any locks when issuing a WTOPC call with PAGE=YES.
- Since the ECB is being suspended during processing of a PAGESTOP, the ECB must NOT be holding any resources (gained through functions such as FIWHC, CORHC, and LOCKC). Holding of resources could result in a lock out condition.
- The WTOPC page control exit point (UOP3) can be used to define the size of a WTOPC output page.

## Examples

- The following example sends a single line output message from E-type code.

```
WTOPC TEXT='DEFINITION NOT ACCEPTED', X
NUM=26,LET=W,PREFIX=VFAC
```

The following example sends a single line output message from CP code.

```
WTOPC TEXT='DEFINITION NOT ACCEPTED', X
NUM=26,LET=W,PREFIX=VFAC,ECB=NO,WKREG=R15
```

These examples will produce:

```
VFAC0026W 13.25.46 DEFINITION NOT ACCEPTED+
```

- The following example sends a single line output message with a substitution field.

```

L R2,VFAINTV
WTOPC TEXT='INTERVAL= SECONDS', X
NUM=4,LET=I,PREFIX=VFAC, X
SUB=(DEC,(R2))

```

This example will produce:

```
VFAC0004I 08.30.00 INTERVAL= 12 SECONDS+
```

- The following example sends a multi-line output message with substitution fields and using the GENMSG and DCTMSG macros.

```

DCTMSG
USING DCTMSG,R3
LA R3,HEADER1 LOAD ADDRESS OF HEADER1 TEXT
WTOPC TEXTA=MSGLEN,PREFIX=DRCT,NUMA=MSGNUM,LETA=MSGLET, X
CHAIN=YES,ENDOFM=NO

WTOPC TEXTA=HEADER1,HEADER=NO, X
CHAIN=YES,ENDOFM=NO

* Loop for each line of text

WTOPC TEXTA=LINETEXT,HEADER=NO, X
SUB=(CHARA,EBW000,CHARA,(R4),CHARA,EBW004,CHARA,EBW008, X
CHARA,EBW012),
CHAIN=YES,ENDOFM=NO

* End of line loop

WTOPC TEXTA=ENDTEXT,HEADER=NO, X
CHAIN=YES,ENDOFM=YES

HEADER1 GENMSG NUM=1,LET=I,TEXT='DISPLAY OF ACTIVE APPLICATIONS'
HEADER2 GENMSG TEXTONLY='APPL CPU SS SSU STATUS'
LINETEXT GENMSG TEXTONLY='....'
ENDTEXT GENMSG TEXTONLY='END OF RCAT DISPLAY' X

```

This example will produce:

```

DRCT0001I 09.58.12 DISPLAY OF ACTIVE APPLICATIONS
APPL CPU SS SSU STATUS
AAAA A BSS HPN ACTIVE
CLGA A BSS HPN ACTIVE
CLGB B BSS HPN ACTIVE
CLGC C BSS HPN ACTIVE
.
.
.
SMPB B BSS HPN ACTIVE
SMPC C BSS HPN ACTIVE
END OF RCAT DISPLAY

```

- The following example sends a multi-line output message from CP code running without an ECB.

```

WTOPC PREFIX=CMKH,NUM=11,LET=E,ECB=NO,ROUT=(PRC,R0), X
TEXT='TERMINATED E-TYPE PROGRAM', X
HEADER=YES,ENDOFM=NO,WKREG=R15

```

\* Loop for output text lines

```

WTOPC ECB=NO,ROUT=(PRC,R0),TEXTA=DECMSG, X
HEADER=NO,ENDOFM=NO,WKREG=R15

```

\* End of text loop

## WTOPC

```
WTOPC ECB=NO,ROUT=(PRC,R0), X
TEXT=' END OF MESSAGE CMKH0011E', X
HEADER=NO,ENDOFM=YES
```

This example will produce:

```
CMKH0011E 10.16.45 TERMINATED E-TYPE PROGRAM
.....message text line.....
.....message text line.....
.....message text line.....
.
.
.....message text line.....
.....message text line.....
END OF MESSAGE CMKH0011E
```

- The following example shows a sample subroutine generates a tabular report with WTOPC PAGE=YES. The program would call this routine for each entry to display, first setting up the display information in the interface areas. When processing is complete, the program should issue

```
WTOPC CHAIN=YES ENDOFM=YES,PAGE=NO,TEXT='DISPLAY COMPLETE'.
```

to indicate the end of a multiple line message. PAGE=NO is used for this last WTOPC to avoid appending 'ENTER ZPAGE TO CONTINUE' to a display that is already complete. This code was taken from one of the command processors.

```
CSM0_REPORT DS 0H
 ST R15,EBW072 SAVE RETURN ADDRESS ACROSS WTOPC
 TM EBW080,CSM0HDR HAS THE HEADER BEEN SENT?
 BO CSM0SBSQ YES, SKIP OVER GEN HEADER
 BAS R15,CSM0_HEADER GO WRITE THE REPORT HEADER
 SPACE 2
CSM0SBSQ DS 0H
*
* WTOPC W/O HEADER AND PREFIX USED FOR ALL SUBSEQUENT DISPLAY
* LINES OF NORMAL DISPLAY
*
 WTOPC TEXTA=(R4),HEADER=NO,TIME=NO,ENDOFM=NO,CHAIN=YES, X
 SUB=(CHARA,RV1NAME,HEX4A,RV1SDA,CHARA,(EBW092,1), X
 CHARA,RV1NETID,CHARA,EBW084,CHARA,EBX084,CHARA,EBX092), X
 PAGE=YES,PAGESTOP=CSM0_EXIT,NEXTPAGE=CSM0_NEWHDR
 L R15,EBW072 RESTORE RETURN ADDRESS
 BR R15 RETURN TO CALLER
 SPACE 2
CSM0_NEWHDR DS 0H
 BAS R15,CSM0_HEADER GO WRITE THE REPORT HEADER
 L R15,EBW072 RESTORE RETURN ADDRESS
 BR R15 RETURN TO CALLER
 EJECT 1
```

This subroutine writes the table header and is referred to in the previous subroutine. WTOPC w/header and prefix is used on the first line of the display. None of these require the NEXTPAGE since this must be the beginning of a display. Each line must be WTOPC'd individually since WTOPC PAGE=YES counts WTOPC's not lines.

```
CSM0_HEADER DS 0H
 ST R15,EBW076 SAVE LINK REG ACROSS WTOPCS
 WTOPC TEXTA=CSM0MHDR,PREFIX=NALS,NUM=01,LET=I,ENDOFM=NO, X
 CHAIN=YES,PAGE=YES,PAGESTOP=CSM0_EXIT
 WTOPC TEXTA=CSM0MS01,HEADER=NO,TIME=NO,ENDOFM=NO, X
 CHAIN=YES,PAGE=YES,PAGESTOP=CSM0_EXIT
 WTOPC TEXTA=CSM0MS02,HEADER=NO,TIME=NO,ENDOFM=NO, X
 CHAIN=YES,PAGE=YES,PAGESTOP=CSM0_EXIT
 OI EBW080,CSM0HDR INDICATE HEADER SENT.
```

```

L R15,EBW076 RESTORE LINK REGISTER
BR R15 RETURN TO CALLER
EJECT 1

* the definitions for the report headers and substitution fields. *

CSM0MHDR EQU *
DC AL1(CSM0MS01-CSM0MHDR-1)
DC C' ' NO TEXT ON PREFIX LINE
CSM0MS01 EQU *
DC AL1(CSM0MS02-CSM0MS01-1)
DC C' NAME SDA STATUS NET-ID CLU-NAME'
DC C' CDRM-NAME TRC'
CSM0MS02 EQU *
DC AL1(CSM0MS03-CSM0MS02-1)
DC C' ---- --- -----'
DC C' ----- ---'
CSM0MS03 EQU *
DC AL1(CSM0MS04-CSM0MS03-1)
DC C'..... INACTIVE'
CSM0MS04 EQU *
DC AL1(CSM0MS10-CSM0MS04-1)
DC C'..... ACTIVE/.. '
DC C' '
CSM0MS10 EQU *
EJECT 1

```

**WTOPC**

---

# Index

## A

- access environment variables 61
- address
  - get file 228, 230
  - get file pool 248
  - pool section file storage 258
- addressing mode characteristic 366
- ALASC macro 28
- allocation
  - changes for storage blocks 314
- ALPHA macro 31
- alphabetic scan macro 31
- alternate resource sharing facility 15
- AMSSC macro 33
- application interface
  - conversion 271
  - macro to select a thread 353
- application name
  - finding 316
- application programming macros xv
- Application System Error macro 378
- assembly language interface 399, 401
- Assign General Tape macro 385
- Assign macros 19
- Assign Unit Record Devices macro 517
- ATTAC macro 34
- Attach a Detached Working Storage Block macro 34
- Attach TPFAR Database Support Structure 134
- attached blocks
  - create an ECB 107
- auto storage blocks 28

## B

- BACKC macro 36
- Backspace General Tape and Wait macro 387
- base register for multiple dsects 130
- BEGIN macro 37
- block 224
  - address
    - validate 528
  - change size 314
  - get storage 248
  - get working storage 244
  - release storage 224
  - reserve storage 303
  - types
    - user-defined 246
  - unhook 512
- block address
  - address
    - get storage 230
- BPKDC 56
- BPKDC macro 40
- BPPSC macro 56
- Build Standardized Scan Tables macro 70

## C

- C language migration 399
- Calculate Record Slot Number macro 268
- calculating an 8-byte file address 193
- CALOC allocation changes 314
- CALOC macro 57
- card reader assignment 517
- CBRW 14, 17, 18
- CC macro 59
- CCIDC macro 60
- central storage dumps 357
- CENVC macro 61
- chain release 336
- change global protect key macro 262
- Change Reserved Storage Block Size macro 314
- characteristics of file addresses 366
- check available system resources 294
- check for request to be completed 531
- CIFRC macro 64
- CINFC macro 66
- CINFC tags 67
- Cipher Program Interface macro 64
- clearing indicators 24
- close a data set 533
- Close a General Tape macro 390
- Close macro 17
- CM0ND macro 70
- CM0PR macro 72
- CMMxxx 67
- CNOSC CHANGE 84
  - return codes 85
- CNOSC DISPLAY 87
  - return codes 89
- CNOSC INITIALIZE 90
  - return codes 92
- CNOSC macro 76
  - CHANGE 84
  - DISPLAY 87
  - general programming considerations 83
  - general syntax 76
  - INITIALIZE 90
  - RESET 95
  - valid return codes 80
  - valid verbs and keywords 76
- CNOSC RESET 95
  - return codes 97
- Combination Data macro 130
- common storage 246, 248
- compute date stamp 132
- Compute Time Stamp macro 395
- Condition Code 59
- condition code settings 18
- conditional core block release 122
- Configuration Constants macro 99
- Configuration Dependent System Equates macro 377
- CONKC macro 99
- contents of macro expansions 4

- control program detected errors 22
- Control Program Interface macro 66
- control program macros 1
- conversation control block entry
  - get 269
- conversion
  - fixed record types 340
  - of record types 340
- convert
  - application interface 271
  - CPUID to processor ordinal number 60
  - FARF address 125
  - LNIATA 333
  - mmchhr address 125
  - NAU 333
  - processor ordinal number to CPUID 60
  - record type macro 340
  - RID 333
  - RVTx 333
  - SCB 333
  - SCBID 333
  - time 397, 504
- convert CPUID to processor ordinal number macro 60
- convert resource application interface macro 271
- convert system ordinal number macro 125
- converting file addresses 191
- core block and file address release 319
- core block reference word 5, 14, 17, 18
- core block release 122
- core block unhook 512
- core resident programs 5
- CORHC 15
- CORHC macro 101
- CORUC 15
- CORUC macro 103
- CP interface 66
- create 12
  - deferred entry 105
  - ECB 338
  - ECB for immediate entry 110
  - ecb with attached core blocks 107
  - low priority deferred entry 120
  - macros 12
  - time-initiated entry 116
- create a deferred entry macro 105
- create a low priority deferred entry macro 120
- create a new ECB for immediate entry macro 110
- Create a New ECB with Attached Core Blocks
  - macro 107
- Create a New synchronous ECB 112
- create a time-initiated entry macro 116
- CREDC macro 105
- CREEC macro 107
- CREMC macro 110
- CRESC macro 112
- CRETC macro 116
- CREXC macro 120
- CRUSA macro 122
- CSMP messages 559
- CSONC macro 125
- CTL dumps 357

CYCPC macro 127

## D

- data
  - field display 138
  - level
    - test and release 122
  - scan macro 349
- data event control block (DECB) 145
- data macros
  - use of 2
- Data Save Area (DSA) 402
- database support structure 136
- database support structure for TPFAR 134
- DATAS macro 130
- date stamp 132
- DATEC macro 132
- DBSAC macro 134
- DBSDC macro 136
- DCTMSG dsect 559
- DDATA macro 138
- DECBC macro 145
- declarative macros 2
- defer processing 149
- defer processing of current entry macro 149
- deferred entries
  - processing 12
- deferred entry creation 105, 120
- deferring processing 16
- Define and Enqueue Resource macro 169
- Define and Hold Resource macro 101
- define event user area 183
- define global fields macro 264
- Define Internal Event macro 181
- DEFRC 16
- DEFRC macro 149
- Delay macro 15
- delay processing of macro 154
- delaying processing 16
- DEQC 15
- DEQC macro 150
- Dequeue from Resource macro 150
- DETAC macro 152
- Detach an ECB Working Storage Block macro 152
- Detach TPFAR Database Support Structure 136
- diagnostic data 357, 363, 378
- diagrams for macro models xvii
- display a control block 552
- Display Tape Status macro 392
- DLAYC 16
- DLAYC macro 154
- DPANL macro 155
- DPROC macro 163
- DSA (Data Save Area) 402
- dsects with single base register 130
- dump
  - snapshot 363
- Dump Facility List Generator macro 291
- dumps 357



duplication  
  determining record 366

## E

### ECB

  creation for entry 110  
  exceptional condition indicators 23  
  program name field 305  
  with attached blocks  
    create 107

Edit and Move Data macro 164

Edit and Send System Message macro 559

EDITA macro 164

end a request 535

end-of-file condition 19

ENQC 15

ENQC macro 169

enqueue resource 169

ENTDC macro 171

Enter a Program and Drop Previous Programs  
  macro 171

Enter a Program with No Return Expected macro 173

Enter Program with Expected Return macro 175

Enter-Back macros 4

enterprise-specific traps, sending SNMP 283

ENTNC macro 173

ENTRC macro 175

### entry

  deferred creation 105  
  delay processing 154  
  get CCB 269  
  get general data set 232  
  low priority deferred creation 120  
  processing complete 190  
  set maximum time 301  
  time-initiated creation 116

### environment list

  access environment variables 61  
    CENVC 61  
  GET/SET/UNSET environment variables 61

environment list variables 61

Epilog for functions that call TPF macro services 399

### equate macros

  use of 2

equates 377

### error

  codes for RELPC macro 332  
  codes in SUD 23  
  indicators for GETPC macro 256  
  macro 357  
  macro for applications 378  
  recording macro 33

### errors

  I/O hardware 22  
  macro parameter 22

EV0BK 183

### event

  destruction 14  
  timeout 14  
  user area definition 183

event facility 14, 187

  wait 187

Event Facility 181

  define 181

  increment count 177

  Status 179

Event Status macro 179

### EVINC

  define user event area 183

EVINC macro 177

EVNQC 14

EVNQC macro 179

EVNTC 14

EVNTC macro 181

EVNWC 14

  define user event area 183

EVNWC macro 187

exceptional condition identification fields 23

### exceptional conditions

  ECB indicators 23

executive macro instruction format 2

executive macros 1

exit control 519

Exit macro 15

EXITC macro 190

expansions of macros

  contents 4

## F

FA4X4C macro 191

FAC8C macro 193

FACE table information 340

FARF address conversion 125

FARW 6, 16, 18

field definitions for globals 264

field scan macro 349

### file 6

  storage allocation macros 6

File a Record macro 195

File a Record with No Release macro 200

File a Single Record macro 203

file address 230

  calculating 193

  characteristics

    symbolic 366

  converting 191

  get 228, 230

  release 319

file address reference word 6, 16, 18

File and Unhold a Record macro 205

File Keyword macro 198

### File macro

  ID errors 22

file macros 9

file pool address

  returning 329

file pool fallback 8

### file record

  referencing 8

file record find 208, 214

- file record find and hold 210
- file record find and wait 216
- file record unhold 510
- File Status Table Information macro 226
- file storage
  - address 252
  - get address 258
  - get large file address 252
  - releasing addresses 6
- file storage address 252
- file type macros 8
- FILEC macro 195
- filing a single record 203
- filing keypoints 266
- FILKW macro 198
- FILNC macro 200
- FILSC macro 203
- FILUC macro 205
- Find a File Record and Wait macro 216
- Find a File Record macro 208
- Find a Single File Record macro 214
- find an RCAT entry macro 316
- Find and Hold a File Record macro 210
- Find and Hold a File Record, and Wait macro 219
- find macro 9
- Find macro
  - ID errors 22
- FINDC macro 208
- FINHC macro 210
- FINIS macro 213
- Finish Program Assembly macro 213
- FINSC macro 214
- FINWC macro 216
- FIWHC macro 219
- fixed record type conversion 340
- FLIPC macro 222
- format for executive macro instructions 2
- FREEC macro 224
- FSTIC macro 226
- function migration 399

## G

- GCFLC macro 228
- GCFSC macro 230
- GDSNC and GDSRC macros 10
- GDSNC macro 232
- GDSRC macro 235
- general data set macros 10
- General File Get File Address macro 311
- general tape assignment 385
- general tape backspace 387
- general tape open 406
- general tape operations 17
- general tape read 484
- general tape record write 492
- general tape reserve 488
- general tape rewind 486
- general use macros xv
- generate a data list macro 237
- generate an ACB 537

- generate an RPL 537
- Generate Message Table for WTOPC macro 242
- GENLC macro 237
- GENMSG macro 242, 559
- get a conversation control block entry macro 269
- get a record 544
- Get an Auto Storage Block macro 28
- Get Core Block and Large File Address macro 228
- GET environment variables 61
- Get File Pool Address and Storage Block macro 248
- get file storage control 260
- get general data set entry macro 232
- Get General Data Set Record macro 235
- get large file storage address macro 252
- get network qualified name (NQN) 271
- Get Program and Lock in Storage macro 254
- get resource definition bytes 271
- get resource identifier (RID) 271
- get session control block identifier (SCBID) 271
- Get Small Core Block and File Address macro 230
- Get Small File Storage Address macro 258
- Get Working Storage Block macro 244
- GETCC block type verification 246
- GETCC macro 244
- GETFC macro 248
- GETLC macro 252
- GETPC macro 254
- GETSC macro 258
- getting storage 57
- GFS control initiate 260
- GFSCC macro 260
- GL0BA keypointing 266
- GL0BY keypointing 266
- GLMOD macro 262
- global field definition 264
- global protect key change 262
- global transaction
  - begin 494
  - commit 496
  - resume 500
  - roll back 498
  - suspend 502
- TXBGC macro 494
- TXCMC macro 496
- TXRBC macro 498
- TXRSC macro 500
- TXSPC macro 502
- GLOBAL1 definition 264
- GLOBAL3 definition 264
- globals 372
- Globals 198
- GLOBZ macro 264
- GLOUC macro 198, 266

## H

- HASHC macro 268
- heap expansion failure 315
- heap storage 57, 224
  - reserving 57
- hold a found file record 210

holding a record 10

## I

I/O associated unusual conditions 22  
I/O hardware errors 22  
I/O operations  
    completion 15  
ID errors on find and file macros 22  
identifying recipients of messages 12  
IGTCCB macro 269  
immediate ECB creation 110  
implied Wait  
    meaning of 6  
Increment Count for Event macro 177  
indicators  
    clearing 24  
initializing 246  
    storage blocks 246  
initializing storage blocks 249  
Initializing Storage macro 57  
Initiate GFS Control macro 260  
input message tokenization support 40  
input/output macros 18  
INQRC macro 271  
Interchange the Status of Two Data Levels macro 222  
interface conversion for application interface 271  
interface for assembler 399, 401  
interface to CP 66  
internal events 187  
Internal Events 14, 179, 181, 306  
Interrogate Event Status macro 179  
invalid format condition 19  
IPSVE macro 275  
IPSVT macro 277  
ITPNT macro 278  
ITRPC macro 283  
IVTYPE macro 246

## K

KARMA macro 285  
keypoint filing 266  
Keypoint updating 198  
KEYRC macro 198, 287  
keyword scan macro 72

## L

large file storage address 252  
Level Test macro 289  
LEVTA macro 289  
library workspace (LWS) 402  
list building 56  
LISTC macro 291  
LNIATA entry location 556  
LNIATA validation 333  
locate terminal entry macro 556  
lock  
    release storage 331  
Lock in Storage macro 254

LODIC 294  
LODIC macro 294  
LONGC macro 301  
look for byte macro 349  
low priority deferred entry creation 120  
LWS (library workspace) 402

## M

macro decoder routine 2  
macro model diagrams xvii  
macro naming conventions 1  
macro parameter errors 22  
macro service epilog 399  
macro service prolog 401  
macro statement  
    sample 6  
macro to file a record 195  
macro usage conventions 3  
main storage  
    getting and releasing 5  
main storage allocation macros 5  
main storage dumps 357  
MALOC allocation changes 314  
MALOC macro 303  
managing data event control blocks 145  
Mark Event Completion macro 306  
MDFB User Attribute Reference Request macro 506  
memory  
    getting and releasing 5  
memory dumps 357  
message  
    route 338  
    send to operator 559  
message tokenization support 40  
message transmission 12  
migration of C functions 399  
mmccchr address conversion 125  
models of macro invocations xvii  
Move program name into specified field 305  
moving a VIPA to another processor 546  
multiple database function (MDBF) macros 24  
multivolume operations 21

## N

naming conventions 1  
NAU validation 333  
network name data  
    locating 353  
network qualified name information 271  
no release file macro 200  
nonreentrant programs 5  
nonzero setting 18  
NQN information 271

## O

Obtain Logical Size macro 362  
Obtain Symbolic File Address Information macro 366  
open a data set 548

- Open a General Tape macro 406
- Open macro 17
- operand formats 2
- Operational Program Zero 13
- operator control of tape operations 20
- OPR dumps 357
- options for error processing 22
- overflow global directory keypointing 266

## P

- parameter formats 2
- PNAMC macro 305
- point for access 550
- pool
  - get file address 248
  - records 6
    - use of 6
  - reordering 6
  - section 6
    - definition of 6
    - file address 329
    - file storage 252
    - file storage address 258
  - storage 228
    - get 228
  - types 6
- pool duration
  - determining 366
- pool section file storage 252
- positional list building 56
- POSTC 14
  - define user event area 183
- POSTC macro 306
- primary global directory keypointing 266
- Print a Line macro 309
- printer assignment 517
- private storage
  - get 244
- PRLNC macro 309
- process selection vector (PSV) table
  - creating entries in 275
  - defining the start and end 277
- IPSVE macro 275
- IPSVT macro 277
- processing of an entry is complete macro 190
- processor ordinal number to CPUID conversion 60
- processor status interface 127
- program macro usage 1
- program migration 399
- Program Record Determination macro 163
- Prolog for ISO-C Functions Calling TPF Macro Services 401
- protect key change for globals 262
- PSMS Utility Interface macro 127

## R

- railroad tracks xvii
- RAISA macro 311
- RALOC macro 314

- ratio dispensing 7
- RCATC macro 316
- RCC 318
- RCC check failures
  - File macro 22
  - Find macro 22
- RCHKA macro 318
- RCPL data
  - locating 353
- RCRFC macro 319
- RCUNC macro 321
- RDCDC macro 323
- Read a Card macro 323
- Read a General Tape Record macro 484
- real-time tape record write 409, 411
- realtime tape operations 16
- record chain release 336
- record code check 8
- Record Code Check macro 318
- record duplication
  - determining 366
- record file and unhold 205
- record file with no release macro 200
- record filing 195
- record find 214
- record find and hold 210
- record find and wait 216
- record hold feature 10
- record scope
  - determining 366
- record size
  - determining 366
- record type characteristic 366
- Record Type Conversion Utility macro 340
- record unhold 510
- reentrant programs 5
- register conventions 1
- REHKA macro 325
- Rehook Core Block macro 325
- RELCC macro 327
- Release Chain macro 336
- Release Core Block and File Address macro 319
- Release Core Block and Unhold File Record macro 321
- Release Core Storage Block macro 327
- release data level 122
- release file record 510
- Release Program from Storage Lock macro 331
- Release Storage Blocks macro 224
- RELFC macro 329
- RELPC error codes 332
- RELPC macro 331
- replace file storage control 260
- Request Keypoint Filing macro 266
- Reserve a Storage Block macro 303
- Reserve General Tape macro 488
- Reserve macros 19
- reserving storage macro 57
- resource allocation 21
- resource application interface conversion 271
- resource definition bytes information 271

- resource ID information 271
- resource sharing facility 15
- Restore Protection Key macro 287
- restricted use macro 1
- Return File Pool Address macro 329
- return to previous program record macro 36
- Rewind General Tape and Wait macro 486
- RID Conversion macro 333
- RID information 271
- RID validation 333
- RIDCC macro 333
- RLCHA macro 336
- ROUTC macro 12, 338
- ROUTC messages 559
- Route a Message macro 338
- routing control application table 316
- RTCUC macro 340
- RVTx validation 333

## S

- sample macro statement 6
- SAWNC macro 345
- scan 31
- Scan Data Field macro 349
- Scan Input Message for Keywords macro 72
- SCANA macro 349
- SCB validation 333
- SCBID information 271
- SCBID validation 333
- SELEC macro 353
- Select a Thread Application Interface macro 353
- self-defining terms 56
- send macros 12
- send system message 559
- SENDC messages 559
- sending SNMP enterprise-specific traps 283
- SERRC macro 357
- SERRC processing 291
- service macros xv
- session control block information 271
- Set Condition Code in Current PSW macro 59
- set entry maximum existence time macro 301
- SET environment variables 61
- sharing storage 246
- Simple Network Management Protocol (SNMP)
  - traps 283
- single file record find 214
- single record filing 203
- SIP macros 3
- SIZBC macro 362
- size
  - obtain logical 362
- SLC Link Alarm macro 285
- small file storage address 258
- SNAPC macro 363
- SNAPC processing 291
- Snapshot Dump Macro 363
- SNMP (Simple Network Management Protocol)
  - traps 283
- SONIC macro 366
- start a program 171, 173, 175
- start of a program 37
- Static Positional List Build macro 56
- status
  - display tape 392
- storage
  - lock 254
- storage address
  - get 228
- storage block initialization 249
- storage blocks
  - attach working storage 34
  - auto storage 28
  - change size 314
  - detach working storage 152
  - filling 246, 249
  - get 228
  - get address 230
  - get common storage 248
  - get working storage 244
  - getting heap storage 57
  - release 224, 327
  - reserve 303
  - validating address 528
  - write a tape record 411
- storage lock
  - release 331
- STP 134, 136
- subsystem local time 395
- SUD error codes 23
- suspend if resources are low 294
- Suspend processing for ECB I/O completion
  - macro 554
- symbolic file address macro 366
- SYNCC macro 372
- synchronize globals macro 372
- Synchronize Tape macro 490
- syntax diagrams xvii
- SYSEQC macro 377
- SYSEQC tag conversion 340
- SYSRA macro 378
- SYSTC macro 381
- system dumps 22
- system error
  - detected by operational programs 22
- system error codes in SUD 23
- system error determination 291, 363
- System Error macro 357
- system error macro for applications 378
- system message
  - edit and send 559
- system ordinal number conversion 125
- system resources 294
- system time-of-day 395
- system utilization 294

## T

- tags for CINFC 67
- tape assignment 385
- tape backspace 387

- tape hardware error condition 19
- tape labeling and multivolume operations 21
- tape macros 16
- tape open 406
- tape operations 20
- tape read 484
- tape record write 409, 411, 492
- tape reserve 488
- tape rewind 486
- tape status display 392
- tape synchronization 490
- TARGET(TPF) migration 399
- Task Communications Area (TSA) 402
- TASNC macro 385
- TBSPC macro 387
- TCA (Task Communications Area) 402
- TCLSC 17
- TCLSC macro 390
- TDSPC macro 392
- Terminal Data Field Display macro 138
- terminal entry locate 556
- Terminal Panel Display macro 155
- Test and Release Data Level macro 122
- Test Sysgen Options macro 381
- thread application interface macro
  - select 353
- Time Conversion macro 397, 504
- time interval for CRETC 13
- time stamp compute 395
- time-initiated ECB creation 116
- TIMEC macro 395
- timeout reset 217
- TMCNA macro 397
- TMSEC macro 399
- TMSPC macro 401
- tokenization support 40
- TOPNC 17
- TOPNC macro 406
- TOURC macro 409
- TOUTC macro 411
- TPF Advanced Program-to-Program Communications (TPF/APPC)
  - ITPNT macro 278
- TPF input message tokenization support macro 40
- TPFAR database support structure 134, 136
- TPFISOC 399, 401
- TPPCC ACTIVATE\_ON\_CONFIRMATION 427
  - return codes 429
- TPPCC ACTIVATE\_ON\_RECEIPT 431
  - return codes 432
- TPPCC ALLOCATE 435
  - return codes 437
- TPPCC CONFIRM 440
  - return codes 441
- TPPCC CONFIRMED 443
  - return codes 444
- TPPCC DEALLOCATE 445
  - return codes 446
- TPPCC FLUSH 449
  - return codes 449
- TPPCC GET\_ATTRIBUTES 451

- TPPCC GET\_ATTRIBUTES *(continued)*
  - return codes 452
- TPPCC GET\_TYPE 454
  - return codes 455
- TPPCC macro 414
  - ACTIVATE\_ON\_CONFIRMATION 427
  - ACTIVATE\_ON\_RECEIPT 431
  - ALLOCATE 435
  - CONFIRM 440
  - CONFIRMED 443
  - DEALLOCATE 445
  - FLUSH 449
  - general programming considerations 426
  - general syntax 414
  - GET\_ATTRIBUTES 451
  - GET\_TYPE 454
  - POST\_ON\_RECEIPT 456
  - PREPARE\_TO\_RECEIVE 459
  - RECEIVE 462
  - REQUEST\_TO\_SEND 468
  - SEND\_DATA 470
  - SEND\_ERROR 474
  - TEST 478
  - valid return codes 421
  - valid verbs and keywords 414
  - WAIT 481
- TPPCC POST\_ON\_RECEIPT 456
  - return codes 457
- TPPCC PREPARE\_TO\_RECEIVE 459
  - return codes 460
- TPPCC RECEIVE 462
  - return codes 464
- TPPCC REQUEST\_TO\_SEND 468
  - return codes 469
- TPPCC SEND\_DATA 470
  - return codes 471
- TPPCC SEND\_ERROR 474
  - return codes 475
- TPPCC TEST 478
  - return codes 479
- TPPCC WAIT 481
  - return codes 482
- TPRDC 18
- TPRDC macro 484
- transaction program name table
  - entries, defining 278
  - ITPNT macro 278
  - search instructions, generating 278
- transfer control to a program 171, 173, 175
- transmitting a message 12
- traps, sending SNMP 283
- TREWC macro 486
- TR SVC macro 488
- TSYNC macro 490
- TWRTC 18
- TWRTC macro 492
- TXBGC macro 494
- TXCMC macro 496
- TXRBC macro 498
- TXRSC macro 500
- TXSPC macro 502



TYCVA macro 504

## U

UATBC macro 506  
UNFRC macro 510  
UNHKA macro 512  
unhold a filed record 205  
Unhold File Record macro 510  
Unhold Resource macro 103  
unholding records 10  
Unhook Core Block macro 512  
unit record control macro 22  
Unit Record Control macro 514  
unit record device assignment 517  
unit record I/O macros 21  
unit record macros 21  
UNSET environment variables 61  
URCTC macro 514  
use of pool records 6  
User Exit Control macro 519  
user traps, sending SNMP 283  
user-defined block types 246  
USURC macro 517  
UXCMC macro 519  
UXMAC macro 523

## V

VALBC macro 528  
validate  
    LNIATA 333  
    NAU 333  
    RID 333  
    RVTx 333  
    SCB 333  
    SCBID 333  
Validate MCHR 366  
Validate Storage Block Address macro 528  
VCHKC macro 531  
VCLSC macro 533  
VENDC macro 535  
VGENC macro 537  
VGETC macro 544  
VIPAC macro 546  
virtual IP address (VIPA)  
    moving to another processor 546  
VOPNC macro 548  
VPNTC macro 550  
VSHOC macro 552

## W

Wait for Event Completion macro 187  
wait for event completion, signal aware macro 345  
wait for request to be completed 531  
Wait macro 15  
WAITC macro 554  
WGTC macro 556  
working storage block 244  
Write a General Tape Record macro 492

Write a Real-Time Tape Record and Release Core  
    Block macro 409  
Write a Real-time Tape Record macro 411  
writing records to tape 16  
wrong length records 19  
WTOPC macro 559

## Z

zero setting 18









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.

SH31-0152-12

