Transaction Processing Facility

Programming Standards

Version 4 Release 1

SH31-0165-10

Transaction Processing Facility

Programming Standards

Version 4 Release 1

SH31-0165-10

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

Eleventh Edition (June 2002)

This is a major revision of, and obsoletes, SH31-0165-09 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.

Contents

Tables

Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

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

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

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

Trademarks

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

AD/Cycle
EOCF/2
IBM
Sysplex Timer
System/390.

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

About This Book

This book presents programming standards used by TPF Development. The standards manual contains information, methods, and examples for implementing user programs compatible with TPF.

This manual contains both statements of standards and examples of standards. The materials used to illustrate standards may not be entirely up-to-date with the current product (for example, the text of an operator message may be changed); they only serve to indicate the points being made. Strict accuracy with a changing product need not be achieved when the intent is solely the purpose of illustration.

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 application and system programmers, and for programming managers.

Conventions Used in the TPF Library

The TPF library uses the following conventions:

Conventions	Examples of Usage
italic	Used for important words and phrases. For example: A *database* is a collection of data. Used to represent variable information. For example: Enter **ZFRST STATUS MODULE** *mod*, where *mod* 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);`

bold italic	Used for emphasis. For example: You ***must*** type this command exactly as shown.
Bold underscore	Used to indicate the default in a list of options. For example: **Keyword=OPTION1** | **DEFAULT**

Conventions	Examples of Usage
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 *pipe* (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=*option*

Scale	Used to indicate the column location of input. The scale begins at column position 1. The plus sign (+) represents increments of 5 and the numerals represent increments of 10 on the scale. The first plus sign (+) represents column position 5; numeral 1 shows column position 10; numeral 2 shows column position 20 and so on. The following example shows the required text and column position for the image clear card.
	`|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...`
	`LOADER  IMAGE CLEAR`
	Notes:
	1. The word LOADER must begin in column 1.
	2. The word IMAGE must begin in column 10.
	3. The word CLEAR must begin in column 16.

Related Information

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

IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF Library Guide*, GH31-0146
- *TPF Application Programming*, SH31-0132
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF General Macros*, SH31-0152
- *TPF Operations*, SH31-0162
- *TPF Main Supervisor Reference*, SH31-0159
- *TPF System Generation*, SH31-0171.

Miscellaneous IBM Books

- *ESA/390 Principles of Operation*, SA22-7201.

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

Introduction to TPF Standards

TPF Programming Standards is for any programmer who writes or modifies programs that interface with the TPF program product. This does not apply to the EOCF/2, TPFDF, or ALCS products, except where explicitly noted.

As used in this document, a programming standard is a method or group of actions that provides a specific solution to a programming problem. Standards and conventions are the interfaces between the TPF development lab and our customers that allow us to limit any disruption to their processor complexes. Most conventions are derived from several years of experience in writing TPF programs. Along with TPF standards and conventions, there are IBM programming standards. All programmers must adhere to items defined as standards when creating or modifying TPF programs.

Note: The word *program* refers to an individual partitioned data set (PDS) member.

Deviations and Additions

Customer requests for additions or changes to this document should be made through your IBM customer representative.

Preexisting Code

While IBM makes every effort to conform to the standards described here, complete adherence is not currently possible because of the nature of the TPF program product. Consequently, some aspects of the TPF program will not conform to these standards. Any deviations from these standards found in TPF 4.1 and earlier releases are considered to be an acceptable, preexisting condition. Preexisting deviations will conform to these standards on a “time available” basis and as business decisions dictate.

Acquired Code

There may be times when IBM will acquire code to be installed as part of the TPF base. Conditions may occur when changing the code to comply with the standards in this document would be difficult or expensive. At the time when the code is acquired, a decision will be made as to whether or not the code will be changed to comply to these standards.

IBM reserves the right to implement acquired code into the TPF base without changing it to adhere to these standards.

Format for I/O Messages to CRAS

This chapter defines the recommended format for all commands (input messages) and output messages. Ensure that all new code conforms to these guidelines. Messages provide a means for the operator to communicate with and to monitor the TPF system. By providing a consistent format for the operator, we are making the job of interpretation and action easier. It is essential not to change the format of existing input and output messages because of the impact on automation.

Commands

Commands (previously known as functional messages), which are sent from the operator to the TPF system, have Z as the primary action code (that is, the first character) and have a 1- to 4-character secondary action code immediately following the Z.

Reserved Secondary Action Codes

The following secondary action codes are reserved:

- Those beginning with U, Z, CTCI, FDRS, and RDRS are reserved for TPF customers.
- The special action code, Z M, is used as an interface to the scrolling package.
- All other secondary action codes are reserved for IBM.

  Within the IBM reserved action codes, the following applies:
  - Secondary action codes beginning with N are reserved for IBM System Network Architecture (SNA) messages.
  - Secondary action codes beginning with L are reserved for IBM line control messages.

Secondary Action Code Format Examples

The secondary action code either ends the message or is followed by a blank.

```
Examples:

ZDTIM             The secondary action code DTIM ends the
                  message.

ZDCOR 000100      DCOR does not end the message, so it is
                  followed by a blank.

ZZDW/0125////     This message format is non-standard.
```

Character Set

For interfaces that receive commands on data level 0 (D0), the character set is restricted to alphanumeric characters A–Z, 0–9, and special characters hyphen (-), period (.), asterisk (*), dollar sign ($), and slash (/).

```
Example:

ZRTCU CTON #KEYPT     This message is nonstandard because it
                      contains a pound sign, #, which is not
                      available on all terminals.
```

Note: Message help functions can use nonstandard characters only if there is also support for an alternative standard form of the message.

```
Example:

ZASER ?             The question mark is a non-standard character;
ZASER HELP          however, the alternative HELP parameter will
                    provide the same function.
```

There is also another interface provided where a copy of the command, as it was entered, is pointed to by field CE2CRSMSG of the entry control block (ECB). This copy of the command message can contain uppercase characters, lowercase characters, other special characters, and is ended by the null character (X'00').

Field Delimiters

Fields in the message should be delimited by blanks. Occasionally, you can use a special character, (hyphen, period, asterisk, dollar sign and virgule) to delimit subfields of a field. All new commands should be free format, that is, all unnecessary blanks in the command should be ignored.

Examples:

ZTINT 280 123456 D1600	Each field is delimited by a blank.
ZDREC 05C.000006 080 006	The record ID field (05C.000006) uses the period (.) to separate the record type from the ordinal number.

Parameters

Commands can contain positional or keyword parameters, or combinations of the two. Positional parameters must be entered in the sequence shown in *TPF Operations*, separated by one or more blanks. Keyword parameters can be entered in any position. The absence of a parameter implies a default specification. The default should be indicated in the command documentation in the individual command specifications.

Commands should support variable length keywords and parameters. The shortest unique character string should be allowed for keywords (that is, the keyword parameters may be truncated to the minimum unique abbreviation). This can be identified in the individual command documentation by uppercase letters. No extraneous data should be entered after the keywords or parameters. If any extraneous data is entered with a message, the message will be flagged as not valid by the message processor.

```
Examples:

ZAUTH D(isplay) LIT-030205   The minimum character representation
                             for display is D.  The second
                             parameter is a keyword.

ZVFAC DIS(play) STA(tus)     The minimum character representation
                             for display is DIS and for status is STA.

ZMPIF DEF(ine) D(evice) N(ame)-$3088G00   The minimum character representation
                                          for define is DEF, for device is D,
                                          and for name is N.
```

Message Parsing Techniques

Use the Input Message Tokenization support (either through the BPKDC macro or the IPRSE C language utility) to edit command parameters. Using these tools saves you the tedium of editing out multiple blanks, checking for an end-of-message, and

so on. When coding new commands, look into the function provided by these macros in *TPF General Macros* and the *TPF C/C++ Language Support User's Guide*.

Messages

Message Generation

Sending most output messages to the system operator is a simple, straightforward, uncomplicated task. For most messages, it is only necessary to point to the output parameters and text, relying on the system to do the formatting. The following macros are provided for this purpose and should be used whenever possible.

WTOPC Edit and send system message

The WTOPC macro constructs a message block and provides facilities for converting binary values into EBCDIC, decimal, or hexadecimal, and for editing the message.

GENMSG Generate message table for WTOPC

The GENMSG macro constructs the information that WTOPC uses to build a message. Multiple GENMSG macros can be assembled together to form message text tables.

DCTMSG Define GENMSG entry

The DCTMSG macro defines and labels entry information generated by the GENMSG macro. These labels can be used when coding a subsequent WTOPC macro.

The WTOPC macro is described in *TPF General Macros* and in the *TPF C/C++ Language Support User's Guide*. GENMSG and DCTMSG are described in more detail in *TPF General Macros*.

Note

If you are coding in C, do not use *printf*. Use the C language *wtopc* macro. See the *TPF C/C++ Language Support User's Guide* for more information.

Character Set

All characters are allowed for messages except the following:

- X'4E' (+), which is the end-of-message control character
- X'6E' (>), which is the start-of-message control character.

TPF system code unconditionally translates these two characters to X'4B' (.). Control program (CP) user exit WTOP is provided for you to customize your output. See *TPF System Installation Support Reference* for more information about this user exit.

Message Format

All messages from the system to the operator, whether reply or unsolicited, have the same basic format:

```
Message_ID Time_stamp Text
```

Message Identification

All messages from the system to the operator, whether reply or unsolicited, must have a unique message identifier (ID). A unique message ID is made up of a segment name, message number, and a severity code in the form *aaaannnns* where:

aaaa Message prefix

For a reply message, this represents the secondary action code of the associated command.

For an unsolicited message, this represents the issuing program or package name.

nnnn Message number

This is a decimal number from 0001 to 9999. Each message should have a unique message number.

s Severity code

This code is used by the operator to determine what additional action, if any, is required. The codes are:

I Information only

This code is appended to all normal response message numbers.

A Action required

This code indicates that additional operator action is required.

W Attention

This code is appended to error messages that may require additional operator action.

E Error

This code is appended to error messages that reflect an error condition without program termination.

T Termination

This code indicates that the program or function is in error and that program termination has taken place.

Message identification is made through the prefix and number only. The severity code is used only as a qualifier.

Message Examples

The following examples show how the message identification guidelines are used.

For command:

```
ZDKAT KPB
```

a normal reply message is:

```
DKAT0001I 11.35.00 KEYPT B  IS PROC UNIQUE AND SS  RESIDENT
                   KEYPT B  FILE ADDRESS 18480013
                   KEYPT B  CORE ADDRESS 000CD458
```

For the command:

```
ZDKAT KPQ
```

an error reply message is:

```
DKAT0006E 11.35.00 INVALID KEYPT NAME, CHOOSE FROM
                   A,B,C,D,E,I,M,V,0,1,2,3,4,5,6,9
```

The following are examples of unsolicited messages:

- CSG50002I 11.45.31 RVT1 RELOADED FROM FILE
- GOGO0001I 11.35.51 APPLICATION FIXED CORE LOADED MODE 7 SSU HPN I-S 3
- CANT0001I 10.25.21 ANT LOAD SUCCESSFUL

Note on Messages for the TPFDF Recoup Package

Message numbers in the 9000–9999 range with the RECP message identification code (ID) are reserved for the TPF Database Facility (TPFDF) product.

Message ID Uniqueness

A message can be identified by more than one ID, but not in the same segment; nor can an ID identify more than one message except where the subsequent message is a more detailed form of previous messages.

Some examples follow:

- The following example is incorrect because the severity code does not make the message ID unique:

  ```
  ABCD0001W THIS IS AN ATTENTION MESSAGE
  ABCD0001E THIS ATTENTION MESSAGE HAS NOW BECOME AN ERROR
  ```

  The message is correct as follows:

  ```
  ABCD0001W THIS IS AN ATTENTION MESSAGE
  ABCD0002E THIS ATTENTION MESSAGE HAS NOW BECOME AN ERROR
  ```

- The following example is correct because the message prefix, (in this case, the issuing program name) makes the message ID unique:

  ```
  ABCD0010E ERROR RETRIEVING RECORD FROM FILE
  EFGH0010E ERROR RETRIEVING RECORD FROM FILE
  ```

- The following example is correct because it is acceptable to reuse a message ID if more detailed information is added to the end of the existing message:

  ```
  ABCD0015E ERROR RETRIEVING RECORD FROM FILE
  ABCD0015E ERROR RETRIEVING RECORD FROM FILE
            BECAUSE RECORD ID WAS CORRUPTED
  ```

- The following example is incorrect because the detailed information has caused the placement of the existing text in the message to change:

  ```
  ABCD0015E ERROR RETRIEVING RECORD FROM FILE
  ABCD0015E ERROR RETRIEVING RECORD nnnn FROM FILE
  ```

Time Stamp

A time stamp is required on all messages. The time stamp must give the value of the 24–hour local standard time clock. (Note that local standard time is subsystem unique. Some messages may require the use of a time-of-day (TOD) clock for time stamping. The time stamp immediately follows the message ID and precedes the message text as follows.

```
MSG ID      TIME STAMP  MESSAGE TEXT
MCHR0050E   10.05.11    INVALID FARF ADDRESS
```

Message Text

There are two kinds of messages: static and variable.

Static Messages

Static messages are those that do not contain variables. There is only one message text and only one meaning.

```
CANT0001I 14.52.24 ANT LOAD SUCCESSFUL
```

Variable Messages

Variable messages contain parameter data. The variable data should not change the meaning or context of a message. There are various kinds of parameter data: main storage displays, tag names, program names, record IDs and numbers, time stamps, addresses, utility names, processor numbers and displays (such as from operator commands ZSTAT, ZDFPC, ZNETW D, and others).

Multiple Line Messages

Output requiring more than one line can be either of two formats: table displays or formatted displays. Each can also be static or variable.

Table displays are usually in tabular form and begin with headings identifying the multiple lines that follow. For example:

```
SYSL0002I 17.52.21
CURRENT SHUTDOWN LEVELS FOR CLASS BATCH

           TOTAL    SHUTDOWN  SHUTDOWN
KEYWORD  ALLOCATED    LEVEL    PCT-AGE
  CMB       258        123       48
  ECB       541        259       48
  FRM      2709       1300       48
  IOB      2272       1090       48
  SWB      1128        541       48
END OF DISPLAY
```

Formatted displays usually have fixed forms and contain the same number of output lines. The output lines can have a mixed style.

```
STAT0008I 14.51.24 SYSTEM STATUS DISPLAY
              IOB  FRAME COMMON    SWB    ECB
ALLOCATED    2720   6000    500   1000   3000
AVAILABLE    2703   5834    465    932   2998

ACTIVE ECBS               2
PROCESSED                 0
LOW SPEED                 0
ROUTED                    0
CREATED                  70
END OF DISPLAY
```

Conventions for Multiple–Line Messages

- All tabular and formatted displays have a header line containing a message ID, time stamp, and explanatory text.
- Any tabular display message must not exceed 64 characters in length. If a tabular display exceeds 64 characters, the column can then be staggered over two lines.

  Note: This does not apply to a single message that wraps.
- The message ID is only found on the first line of a message. It is not repeated on subsequent lines.
- The last line of a multiple-line display must be one of the following:

  - END OF DISPLAY
  - END OF MESSAGE
  - A “continuation” line.

  It is suggested that you use the ZPAGE function to display continuation messages. Remember that subsequent pages of a multiple–line message that are displayed after ZPAGE has been entered are considered new messages. Therefore, a header line containing the original message ID should be the first line of subsequent pages.
- Variable displays do not have null results; instead, they result in a separate message indicating that there is nothing to display.
- If an error occurs while successive lines in a multiple-line message are being displayed, output from the error message is displayed only after an END OF DISPLAY line has been sent. This separates the multiple-line message from the error message.
- Multiple-line messages consist of multiple blocks, not multiple messages.

Formatted displays that contain no variable data are simply static displays.

```
MPIF0001I 14.51.24 MPIF HELP DISPLAY
TRACE - USED TO TRACE ACTIVITY OF A USER, PATH OR DEVICE
 USER    - NAME OF THE USER
 PATH    - NAME OF THE PATH OR DEVICE
 ALL     - TRACE ACTIVITY ON ALL PATHS
 STOP    - STOP ACTIVE TRACE
 DISPLAY - DISPLAY THE IN-CORE MSRB TRACE TABLE
 NOLOG   - DO NOT WRITE MSRB TRACE TO REAL-TIME DUMP TAPE
END OF DISPLAY
```

Modifying Existing Messages

The following modifications are allowed to existing message text because they will not cause breakage to existing automation procedures that follow the recommended message parsing techniques that are described in Message Parsing Techniques on page 10:

- Message text with no substitutable data fields may be changed as necessary provided the meaning of the message text does not change. A substitutable data field (or fields) may be added to the message text, however, any subsequent change must follow the next rule.
- Message text with substitutable data fields may be changed as long as the number of tokens that are needed to represent the message text and the position of tokens representing substitutable data fields in the message text does not change. In addition, the following changes may be made to these messages:
  - Additional values may be added to an existing substitutable data field, such as a new value for *rc*. For example:

    ```
    TRAC0001I  TRACE MODIFY COMPLETE RC=rc
    ```

    If a human readable text is used to represent the values of *rc*, each value must be represented by the same number of tokens.
  - The length of substitutable data fields may change.
  - New substitutable data fields may be added to the end of the message text provided the meaning of the original message does not change.

Defining New Messages

When a new message is being considered for a segment, examine all existing messages that are issued by that segment to determine whether an applicable message already exists. If no message text exists that satisfies the requirements, a new message can be generated.

If message text exists in a segment, it must be used with the existing message ID. There should be no duplicate message texts in a segment.

Message text must depict only one meaning. If a message already exists for one purpose, but its text could be interpreted differently for another condition, a different message and message ID must be used.

Message Parsing Techniques

Message parsing becomes extremely important in the context of automation. Console automation and test script automation (with a product such as EOCF/2) are the most relevant for the TPF system. The method of automation influences the parsing techniques used.

- Parsing the Message ID

  Techniques for parsing the message ID should be structured in such a way that they will be independent of system changes.

  In EOCF/2, the message filter parses the message ID. The selected message is then passed to a user automation script. The automation script does not need to parse the ID.

  The solution for parsing a message ID is to break the ID into its parts and to compare each part against the same part of each ID in a predefined set. The prefix and severity code can be compared by using string compares, while the number can be compared using a numeric comparison. This numeric comparison will ignore any leading zeros, and therefore, will allow the serial number to be expanded without the threat of reducing the effectiveness of the automation.
- Parsing variables

  Parsing variables is only necessary for variable messages. Static messages are always identified by the message ID.

  Either console or test script automation will have to parse the message text.

  Always parse the text by using tokens (strings delimited by blanks and special characters), not by position. Tokens allow easier handling of variable length fields.

System Errors

Control programs and real-time programs can execute the SERRC macro or the SNAPC macro to issue a system error. However, use the SERRC macro with discretion because of its high system overhead. With the size of main storage increasing, criteria for issuing system errors has become more rigorous. The SNAPC macro produces more manageable dumps without the overhead required by SERRC.

If a problem can be adequately signalled without a dump, do not take a dump. If a smaller, ″snapshot″ of memory is enough, it should be taken in preference to a complete dump. The SNAPC macro is preferred for such a snapshot. If the SERRC macro is used, the SLIST parameter lets you define (and restrict) storage lists to be dumped.

Note: During SERRC macro execution, the CPU becomes dedicated to dumping areas of main storage to the real-time logging/activity tape. The dump time is measured in seconds, and during this time the CPU cannot do productive work. This is not true for a SNAPC dump.

The SERRC macro also creates an entry control block (ECB) to send a system error message to the computer operator, and an ECB to carry out a keypoint update.

The following system error numbers are reserved:

- For the CTCI package: A20000–A20FFF using prefix Z.

Record IDs

This chapter discusses database record identifiers (IDs) and tape record IDs.

Database Record IDs

All new database record IDs used by the TPF system will fall in the following ranges:

- X'0000' to X'00FF'
- X'FC00' to X'FFFF'

Current record IDs that fall outside this range will not be changed at this time to comply with this standard. See Appendix B, “Existing Database Record ID Deviations” on page 33 for a list of existing database record IDs which deviate from this standard.

Tape Record IDs

All new tape record IDs that are used by the TPF system will fall in the following ranges:

- X'EA00' to X'EFFF'

Current record IDs which fall outside this range will not be changed at this time to comply with this standard. See Appendix C, “Record IDs Used for RTA/RTL Tape” on page 35 for a list of existing tape record IDs that deviate from this standard.

Naming Conventions

This chapter contains naming conventions for E-type programs, ISO-C build scripts, ISO-C library interface scripts, macros, coupling facility (CF) list structures, and other names.

Source File Names

The following table identifies how to name source files.

Table 1. Naming Conventions for Source Files

Type of Source File	Naming Convention	Reserved Names	Notes
CP CSECT	CC*xxxx*, where *x* is any alphanumeric character.	See Appendix A, “Deviations from Existing Naming Standards” on page 29 for a list of CP CSECTs that deviate from the naming conventions.	
COPY members	*axxx*|*x*|*x*, where *a* is B, C, D, U, JC, or XL, and *x* is any alphanumeric character. The maximum length is six characters.	See Appendix A, “Deviations from Existing Naming Standards” on page 29 for a list of copy members of CP CSECTs that deviate from the naming conventions.	
E-type (real-time) program • load module (DLM, library, DLL) • BAL • TARGET(TPF) **Note:** The program name is the name in the TPFLDR input deck.	*axxx*, where *a* is B, C, D, U, JC, or XL, and *x* is any alphanumeric character.	• For the TPF Database Facility (TPFDF) product: members starting with BCB, BFA–BFB, BRC, BRZ, and all members beginning with UF (**except** those that start with UFF and UFH–UFJ, which are reserved for the TPFDF product for atraxis (Swissair) only). • For the Step-by-Step trace (SST) facility product: members starting with BT (BT*xx*). • For the CMS/TPF product: members in the CTP0–CTP9 and CTPA–CTPZ range. • For atraxis (Swissair): members starting with DC (DC*xx*), DI (DI*xx*), DO (DO*xx*), DP (DP*xx*), and CVZK, CVZN, and CVZO.	• BRCP and BRCQ are already in use by the TPF product and are not available to TPFDF. • BTCB, BTIM, and BTLI are already in use by the TPF product and are not available to the SST facility. • See Appendix A, “Deviations from Existing Naming Standards” on page 29 for a list of those TPF real-time programs that deviate from the naming conventions.
C/C++ source	*axxx*/*x*/*x*, where *a* is B, C, D, U, JC, or XL, and *x* is any alphanumeric character.		• The standard is to have 4- to 6-character names, but the guideline is to use 5- to 6-character names.[1]
C/C++ build script	*axxx*BS • where *a* is B, C, D, U, JC, or XL, and *x* is any alphanumeric character. • *axxx* must match the load module name.	Because *axxx* matches the load module name, the same list of reserved names for E-type (real-time) programs applies.	
C run-time (nondynamic) library interface script	*axxx*XV • where *a* is B, C, D, U, JC, or XL, and *x* is any alphanumeric character. • *axxx* must match the load module library name.	Because *axxx* matches the load module name, the same list of reserved names for E-type (real-time) programs applies.	
Ported code segments	Same as the ported code name, unless it is a system-required function that must go through SIP. If you must change the file name, you must follow the TPF standard for file names.		• Ported code segment names can be reused as long as the type is different. ABC.C and ABC.H are allowed. The key is that no two executable programs can have the same name. • Six-character segment names will be allowed for ported code. • Ported code will not have SID codes automatically placed on changed lines.

Table 1. Naming Conventions for Source Files (continued)

Type of Source File	Naming Convention	Reserved Names	Notes
TPF external C/C++ header (TPF interface protected; interface guaranteed)	C$*xxxx*, where *x* is any alphanumeric character.		• If an equivalent BAL DSECT already exists, *xxxx* should be as close as possible to the name of the BAL DSECT. • If there is no BAL DSECT equivalent, there is no restriction. However, if an equivalent BAL DSECT is created, *xxxx* must match for both the external C/C++ header and BAL DSECT.[2] • Existing C headers, including those for drivers, will **not** be renamed and will remain exceptions.
TPF internal C/C++ header (TPF implementation specific; interface not guaranteed)	I$*xxxx*, where *x* is any alphanumeric character.		• If an equivalent BAL DSECT already exists, *xxxx* should be as close as possible to the name of the BAL DSECT. • If there is no BAL DSECT equivalent, there is no restriction. However, if an equivalent BAL DSECT is created, *xxxx* must match for both the internal C/C++ header and BAL DSECT.[2]
Ported C/C++ header	Same name as ported header.		Ported C/C++ header files will not be renamed and will be named “as ported” unless they are more than 8 characters long. Ported C/C++ header files are restricted to 8 characters in length. Ported C/C++ header files are exceptions to the C/C++ header naming conventions.
Standard header	Same name as standard.		
BAL imperative macro (any macro that generates code)	*a*(*xxxx*)C, where *a* is an alphabetic character and *x* is any alphanumeric character. Note: *xxxx* is variable in length and can be from 1–4 characters in length. BAL imperative macros can be no longer than 6 characters in length but can be fewer than 6 characters.	• For the Step-by-Step trace (SST) facility product: SST*xx*C	• Does not apply to system generation macros. • See “Macros” on page 29 for a list of existing macros that currently deviate from these naming standards. • The FDRSC macro name is reserved for customer use.
BAL equate, declarative, keypoint, and control block macros [3]	I*xxxx*(*x*), where *x* is any alphanumeric character. **Note:** The 5th *x* is obsolete. All existing BAL equate, declarative, keypoint, and control block macros that are 6 characters in length are exceptions and will not be renamed.	• For the Step-by-Step Trace (SST) product: ISST*xx*. • For atraxis (Swissair): – ID*xx*DC – IR*xx*DC – IT*xx*DC – IW*xx*DC	• Field names **must** start with **I**. • Does not apply to system generation macros. • If a BAL DSECT is created and it will also have an equivalent C header, the BAL DSECT will be I*xxxx*, – where *xxxx* is any alphanumeric character. – where *xxxx* must match for both the BAL DSECT and C header.[2] • See “Macros” on page 29 for a list of existing macros that currently deviate from these naming standards.
BAL global set symbols in macros	&*xx*, where *x* is any alphanumeric character.	All global set symbol names starting with &SU, &XU, and &U are for customer use.	Global symbols must be unique to the system.

Table 1. Naming Conventions for Source Files (continued)

Type of Source File	Naming Convention	Reserved Names	Notes

Notes:

[1] The PDS member name for the compiled or assembled object code will be the same as the source file member name. This name must be unique in the directory that it is in. See *TPF Application Programming* for details about ISO-C, build scripts, and the library interface tool. See the BSCRLSCR prolog for a sample build script format.

[2] In the following example, a new C header file is created and no BAL DSECT exists:

```
C$TABL (external header)

struct tpf_itabl_item
{
  char itabl_field1;
  short int itabl_field2;
}
```

But then, if a C$TABL BAL equivalent is created, it would be named ITABL. For example:

```
ITABL
  ITABL_FIELD1   DS   CL1
  ITABL_FIELD2   DS   XL2
```

In the following example, a new C header file is created and there is an existing BAL DSECT:

```
ECBHDR      a "made-up existing" TPF BAL DSECT
C$ECBH      /* its new "made up" C language header

            The field name in the C header should (but is not required to)
            match those in the BAL DSECT.
```

[3] Any equates that are specific to a declarative macro will be defined in the macro definition. For example, if the macro has indicator bytes with predefined bits, those bits will be defined via equates in the macro. These equates are available for program use with the macro call. The naming convention for all labels in a macro (including equates) should follow the macro naming convention and also begin with the letter **I**.

Contents of C/C++ Source Files

The following table describes how to name the *C/C++ item* that is in a C or C++ source file; these are the actual contents, not the external source file name. These conventions are in place to avoid clashing with namespaces of customers:

- If a header is included in and compiled with customer source code.
- If external symbols are generated (for example, the names of ISO-C library function stubs).

Table 2. Naming Conventions for C/C++ Source Code (Contents of File)

Item in C/C++ Source File	**Standard and Ported C/C++ Code**	**TPF - Interface Protected (Interface Guaranteed)**	**TPF - Implementation Specific (Interface Not Guaranteed)**	**BAL Equivalent Exists**
identifiers • functions[1] • data macro (#define) typedef enumeration • tag • value tags[5] • structure[6] • union exported • functions • data	As is[2]	One of the following[3]: • tpf_ • TPF_ • tpf*xxx*_ • TPF*xxx*_ where *xxx* is a package or feature code that will allow the names to be grouped together in documentation and also self-document to which package they belong.	Anything	One of the following:[4] • TPF_name • TPF_name_*xxx* • tpf_name • tpf_name_*xxx*
names coded with the `#pragma map` directive or assembler external symbols **Note:** Names beginning with @@*LM* are reserved for TPF link map support for C load modules.	@@*xxxxxx*, where *x* is any alphanumeric character. **Note:** *xxxxxx* is variable in length and can be from 1–6 characters.	@@*xxxxxx*, where *x* is any alphanumeric character. **Note:** *xxxxxx* is variable in length and can be from 1–6 characters.	@@*xxxxxx*, where *x* is any alphanumeric character. **Note:** *xxxxxx* is variable in length and can be from 1–6 characters.	@@*xxxxxx*, where *x* is any alphanumeric character. **Note:** *xxxxxx* is variable in length and can be from 1–6 characters.

Table Notes:

1. For external function names, the name in the header file must map as *@@nnnnnn* for a library function and *Cnnn* for a dynamic load module (DLM). For example:

```
             Name in Header File
             -------------------
                                 pragma maps
    library  tpf_sort()            -------->  @@SORT
                                       to

                                 pragma maps
    DLM      tpf_cima_sort()       -------->  CIMA
                                       to
```

   See “Mapping Library Names and LONGNAME Support” on page 25 for information about LONGNAME support.

2. If you are porting a standard package but find that you need to write TPF-unique function that will still be considered part of the standard package, use the naming convention of that package and add a suffix of _tpf. Following is an example with a standard package of `standpkg` and a function of `newfunc`:

```
standpkg_portedfunc   /* Part of the package and ported as is   */
standpkg_newfunc_tpf  /* Part of the package but written by TPF */
```

3. TO2_ functions are exceptions to this naming convention. They will not be renamed.
4. All existing TPF DSECTs will not be renamed and are exceptions.
5. Global Tags and System Equates Migrated from Assembler to ISO-C:
   - Use an underscore character followed by the name in lowercase when naming global variables. For example, global assembler tag *@globz* would be *_globz* in C. Except for the underscore (_) substitution, names should follow the existing assembler standard.
   - Use uppercase characters when naming constants. Replace the # at the start of some system constants with an underscore character (_). Comment on the use or purpose of all constant values.

```
The following assembler system equates

#SYSEQ    EQU     12       maximum number of jobs allowed in system
WIGITCNT  EQU     42       number of wigits supported

become, in C

#define _SYSEQ    12    /* maximum number of jobs allowed in system */
#define WIGITCNT  42    /* number of wigits supported               */
```

6. If a C or C++ header is created for an existing BAL DSECT, prefix the C or C++ structure name with TPF_ or tpf_. If there is an existing BAL DSECT, member names in the C or C++ structure are the same as in the corresponding BAL DSECT.

Generally Reserved Names for the TPF System

The letters I, i, TPF, TPF_, tpf, and tpf_ are reserved for future use by IBM. This standard applies to the following names.

- TPF file system file names
- TPF collection support data store (DS) names
- Event names created by EVNTC and ENQC macros
- DECB namespaces

- Coupling facility (CF) structure names
- Logical record cache names.

TPF File System File Names

The TPF file system file names reference links to file system objects. These objects include:

- Directories
- Regular files
- Special files
- Symbolic links.

You can access any file system object other than `tmpfiles` by name. The file system file names have as many as 256 characters and contain a hierarchical directory structure that is largely compatible with the Portable Operating System Interface for Computer Environments (POSIX) standards, although this directory structure does not contain executable files.

Note: The TPF system does not implement the POSIX standards, and the TPF file system application programming interface (API) functions are not POSIX compliant, although they are modeled after the POSIX standards and, for the most part, are POSIX compatible.

TPF Collection Support Data Store Names

A TPF collection support data store (DS) is a named set of collections residing in a TPF subsystem. DS names are limited to 8 characters.

Symbolic Register Names

Register naming conventions for TPF system programs now require the use of the equated values R0–R15. See the following table.

New Register Names	Old Register Names
R0	RAC, RG0, R00
R1	RG1, R01
R2	RGA, RG2, R02
R3	RGB, RG3, R03
R4	RGC, RG4, R04
R5	RGD, RG5, R05
R6	RGE, RG6, R06
R7	RGF, RG7, R07
R8	RAP, RG8, R08
R9	REB, RG9, R09
R10	RLA
R11	RLB
R12	RLC
R13	RLD
R14	RDA

R15	RDB

Coupling Facility Structure Names

There are two types of coupling facility (CF) structures:

- *List structure*, which is a named piece of storage on a CF that enables users to share information organized as entries on a set of lists or queues. A *user* is an application or an instance of an application using connection services to access a CF list structure. Because users ***connect*** to a CF list structure to access it, users are also referred to as *connections* or *connectors*.
- *Cache structure*, which is a named piece of storage on a CF that enables users to share information. A CF cache structure allows high-performance sharing of frequently referenced data.

Logical Record Cache Names

A *logical record cache* provides high-speed access to data, which enables you to develop data sharing programs with improved performance. You can use a logical record cache for data consistency, which ensures the validity of the data that is shared and keeps track of data that resides in permanent storage and the local cache.

Tape Names

IBM reserves all tape names that have a numeric value (0 - 9) as the second character; for example, A4X or B2B.

Recoup Descriptors

The following recoup descriptors are reserved for customer use:

- BKW*x*
- BKX*x*
- BKY*x*
- BKZ*x*

C/C++ Standards for the TPF System

This chapter briefly discusses the C and C++ standards for the TPF system.

Compile Options for TPF Programs

All ported C code that requires TPF platform-specific code will be included or excluded during compilation with _TPF.

You must compile programs the way that they are defined in SPPGML and with the defaults that are set by the multiple assembler (MASM).

Mapping Library Names and LONGNAME Support

Library functions must have an associated `#pragma map`. Prefix the mapped name with @@. Use the remaining 6 characters to indicate the function that is being mapped. For example, the `longjmp` function is mapped as follows:

```
#ifndef _TARGET_TPF
#pragma map(longjmp,"@@LONGJM")
#endif
```

Structure and DSECT Definition and Documentation

IBM recommends that you use packed format only for existing structures for compatibility with assembler code. Allowing the compiler to set the alignment of structures helps to improve performance.

Using the CHECKOUT Option for C Headers

The IBM C compiler products on the System/390 platform have a `CHECKOUT` option that provides warnings for dubious code and provides informational messages as warnings. All C header files must compile cleanly with the `CHECKOUT` option, while all C language executable code must compile cleanly without the `CHECKOUT` option.

External headers must compile with RC=0 with the `CHECKOUT` option.

Use the following parameters to compile with the `CHECKOUT` option:

```
CHECKOUT(NOPPC)

This works for the IBM SAA AD/Cycle C/370 Compiler.
```

Note: Depending on the compiler, the parameters can change at a future time.

Using the C++ INFO Compiler Option for C Headers

The IBM C++ compiler products on the System/390 platform have an `INFO` option that will generate warning and informational messages. All C header files must compile cleanly with the C++ compiler `INFO` option as well as with the C compiler `CHECKOUT` option.

External headers must compile with RC=0 with the `INFO` option.

Use the following parameters to compile with the `INFO` option:

```
INFO(NOPPC)
```

Note: Depending on the compiler, the parameters can change at a future time.

Assembler Coding Practices

This chapter provides some recommendations for coding in assembler language.

Register Usage

The following gives the TPF system perspective on register usage, but not the application perspective. Registers R0–R7 are saved for an application by the TPF system.

Registers Reserved for the Application Program Interface

Register 8 = R8 =	TYPE E PROGRAM BASE REGISTER
Register 9 = R9 =	ENTRY CONTROL BLOCK BASE REGISTER

Registers Reserved for Control Program Use

Register 11 = R11 =	CONTROL PROGRAM BASE REGISTER (fixed)
Register 12 = R12 =	CONTROL PROGRAM BASE REGISTER (fixed)
Register 13 = R13 =	CONTROL PROGRAM STACK REGISTER

Registers R10, R11, R12, R13, R14, and R15 are not guaranteed across macro calls.

Program Structure

Type E assembler programs must contain a standard prolog and use the following format:

- BEGIN macro
- Copyright statement
- General prolog
- FINIS macro.

The FINIS macro, together with the BEGIN macro, calculates the program size. This calculation is required by the system loader to ensure that the program is not greater than its allocated size. The FINIS macro also generates an LTORG statement, which causes all literals to be generated before the FINIS macro.

Use of Global Variables for CSECT Statements

When you complete a DSECT definition, resume the previous control section (whether CSECT or DSECT). The method used to resume the previous control section depends on the mechanism that is used to define the DSECT:

- If the DSECT is defined inline in a CP segment, use `&CONPRO CSECT` to resume the CP CSECT.
- If the DSECT is defined inline in an assembler real-time segment, use `$IS$ CSECT` to resume the real-time CSECT.
- If the DSECT is defined inline in an assembler object file included in an ISO-C load module, use `&CG2 CSECT` to resume the C load module CSECT.

  Note: This requires you to have explicitly included a `GBLC &CG2` statement in the C load module.

- If the DSECT is defined in a macro definition, use `&SYSECT &SYSSTYP` in the macro definition to resume the control section that was in effect when the macro was called.

VCONC Macro

Do not directly generate LEDT-resolved V-Type constants (VCONs). Call the VCONC macro to generate them, because this macro handles both assembler and ISO-C VCONs correctly. Using VCONC allows both ISO-C and BAL programs to call BAL macros and handles the different externals generated by both.

Appendix A. Deviations from Existing Naming Standards

This section lists the deviations from the IBM naming standards previously defined for the following real-time (E-type), control program (CP), macro, and C/C++ language header file names.

TPF Real-time Segments

ACPD	ACPE	ACPF	ACPJ	AMX2	ARDW	ARDY	ARD3	ARD5
ARD6	ARD7	ARD8	ARD9	ARPC	ARPI	ARPK	ARPT	ARP1
ARP2	ASCA	ASC1	ASC2	ASC3	ASC4	ASC5	ASC6	ASC8
ASFI	ASFJ	ASFO	ASFP	ASF1	ASF2	ASF3	ASF4	ASF5
ASF6	ASF7	ASF8	ASF9	ASL1	ASL2	ASL3	ASL4	ASL5
ASL6								

FMSG	FMS2	FMS3	GLBL	GOGO	GOG1	GOG3	GOG5	PTVB
PTVC	PTVE	PTVF	PTVG	PTVH	PTVK	PTVL	PTVP	PTVR
PTVS	PTVT	PTVV	PTVX	PTV0	PTV1	PTV2	PTV3	PTV4
PTV5	PTV6	RCTD	RLCH	WGA1	WGRA	WGR1	XHAA	XHA1
XHA2	XHA3	XHA4	XHBB	XHB1	XHCC	XHDD	XHEE	XHFF
XHGG	XHHJ	XHHK	XIAA	XIA1	XIBB	XICC	XIDD	XIEE
XNAA	XNBB	XNEE	XNRB	XOAA	XOBB	XOCC	XODD	XTAA
XTA1	XTBB	XXAA	XXA1	XXBB	XXCC	XXDD	XXEE	XXFF
XXGG	XXHH							

CP CSECTS and Copy Members

The following CP CSECTS and copy members deviate from the naming standards.

CPLKMP	EPTV	FPTV	GPTV	GRFS	HPTV	IBF1	IBF2	IB01
IB02	IB03	IB04	IB05	IB06	IB07	IB08	IB09	IB10
IBF1	IPL2	IPTV	JPTV	KPTV	MPTV	NPTV	RTCU	

Macros

The following imperative-type macros deviate from the naming standard of ending with the letter C. For more information about imperative-type macro naming standards, see Table 1 on page 16.

$AOR	$CNFMD	$CNFRM	$FLUSH	$GETAT	$GETYP	$IRSTR
$POSTR	$PTRCV	$RECVW	$RTSND	$SENDD	$SENDE	$TEST
$WAIT	ALPHA	BBEWP	BBWRT	BCRUS	BEGIN	BFIND
BLOAD	BRSTR	BSAVE	BTEST	CASE	CM0ND	CM0PR
CONVT	CPRND	CRUSA	CTKL	CVRTK	DCL	DCLREG
DDATA	DIAG	DO	DPANL	EDITA	ELSE	ENDDO
ENDIF	ENDSEL	FILKW	FINIS	FONTA	GCALX	GCLKX
GDATX	GENMSG	GFMTB	GFMTD	GLMOD	GLOBZ	GOTO
ICALL	ICELOG	ICPLOG	IDATB	IDATG	IDOTB	IF
ILCKCB	INDEX	IPSVE	IPSVT	ISKLG	ISNSE	ITPNT
IULKCB	IVTYPE	JRET	KARMA	LEAVE	LET	LEVTA
MPGMA	MPY	OLDTBL	OTHERW	PKSTG	PM0CN	PTVERM
RAISA	RCHKA	REHKA	RIATA	RITID	RLCHA	SCANA

SCASE	SELECT	SENDG	SET	SKLNG	SNAKEY	SPMEQ
SSKE	SYSRA	TAPMA	THEN	TMCNA	TPFGLB	TYCVA
UDATB	UNHKA	UNTIL	WHEN	WHILE		

The following is a list of data macro (DSECT) names that deviate from the naming standard of beginning with the letter I. For more information about declarative macro naming standards, see Table 1 on page 16.

AD0DS	AD1WB	AD2IN	AM0SG	AN0NT	AO1ON	AR0RT
AS0MP	AS1MP	AS2MT	AS4MF	AS5MB	BCPBC	BC0SA
BK0DF	BK0LC	BK0RP	BK0UX	BK1RP	BL0RF	BL0RP
BMGLA	BMGLB	BMGLC	BMGLD	BMGLE	BMGLF	BMGLG
BMGLP	BMGLQ	BMGLY	BR0ID	BR1ID	BS0AT	BXLBC
CA4CT	CB8HD	CB9PD	CCPTB	CD0BT	CD0DC	CD0DT
CGNDS	CI0CO	CI1AP	CJID	CJ3TP	CJ6KP	CK0KE
CK1KE	CK2KC	CK2SN	CK6KE	CK7KE	CK8KE	CMAMC
CMFEQ	CM0MC	CM8CM	CN0CM	CN0CN	CN0CP	CN1ST
CO1DR	CO3NL	CPTIC	CP0SK	CR0AT	CS0CA	CT0SD
CV0CM	CW0CC	CW0CW	CX#EV	CX7CW	CX8CW	CY$CR
CY0PD	CY1KR	CY2KT	CY3DR	CY5GT	CY5PA	CY6TR
CY7PL	CY8TB	CY8WB	CY9KP	CZ1GF	CZ1UX	C404C
C50C5	DBRREC	DB0DB	DC0DP	DCTBCR	DCTBPK	DCTBRD
DCTBXP	DCTCDB	DCTCIO	DCTCRW	DCTCTL	DCTCWA	DCTDCL
DCTDCR	DCTDDB	DCTDNT	DCTDOR	DCTECH	DCTERI	DCTFCA
DCTGDS	DCTGEN	DCTGFN	DCTICD	DCTIGT	DCTIRB	DCTISB
DCTIST	DCTISV	DCTMAT	DCTMCH	DCTMDR	DCTMFS	DCTMGT
DCTMIO	DCTMPR	DCTMRB	DCTMSG	DCTMUP	DCTMUS	DCTOLC
DCTOLD	DCTOLI	DCTOLM	DCTORB	DCTPAN	DCTPDT	DCTPFX
DCTPTV	DCTRDF	DCTRFN	DCTRIT	DCTRTX	DCTSCH	DCTSCT
DCTSOI	DCTSON	DCTSTK	DCTSWC	DCTTIO	DCTTOK	DCTTRC
DCTUCL	DCTVAL	DCTVFA	DC0DC	DC0RT	DR0IL	DR1IL
EB0CN	EB0EB	EB1PR	EP0EC	ER1ER	ES4ES	EV0BK
E80E8	FC0TB	FI0CB	FN1FN	FR0RT	GENFD	GL0AT
GL0BA	GL0BB	GL0BC	GL0BD	GL0BE	GL0BF	GL0BG
GL0BP	GL0BQ	GL0BY	GO1GO	GROUP	I80I8	I82I8
LCODS	LDCRL	LDEVBK	LK4KC	LK5KC	LK6KC	LOCORE
LTDRT	L80L8	MA0TB	MC0MC	MI0MI	MK0CK	MPRECP
MS0AT	MS0UT	MT0MT	NA0AT	NC0CB	NC2EC	NF1NF
NP0CP	OB0IR	OB1IR	OL1OL	NC2EC		
PI1DT	POSTPT	PO1PO	PQ5CQ	PR1OT	QWEQU	RC0AT
RC0EQ	RC0PL	RC1IT	RECOUP	RGEQUC	RR0RT	RTTEQ
RUNID	RV0VT	RV1VT	RV2VT	SA0AT	SCKDS	SC0TM
SDFPF	SD0RV	SE0MS	SH0LL	SI0GT	SI1GT	SI3CT
SI3GT	SI4CT	SI5CT	SLSTL	SN0CT	SP0KY	SP0PA
SRCK1P	SRHH1P	SRM31A	SRM41A	SRM51A	SRM61A	SR54BA
SR0RT	SS0OR	STDHD	ST0TB	ST0TM	ST1ST	TA0PP
TC0TS	TE0TE	TI0ME	TI0MP	TI0MT	TI1TI	TO9TO
TPLDR	TPPCE	TS0TS	TVDSA	TVDSB	UA1UA	UCNFEQ
UD0RV	UI0OM	UI1OM	UI2PF	UI3MP	UR0IO	UR1DS
UR1ST	UT2RT	UU1TT	UV1RP	UV3RP	UW2CP	UX1DQ
UX1PL	UY1BQ	UZ1PQ	VF0AC	VK4CK	VSFLWA	WA0AA
WA1AA	WG0TA	WI2BS	WI3BS	WI4BS	WI5BS	WI6BS
WO0RK	XA1DS	XB0XB	XB1XB	XC1CC	XD0LS	XE1SC
XF1FF	XH0XH	XI0DS	XI1XI	XJ1LC	XK1CT	XL0DS
XM0RL	XM5XM	XN1XN	XP1XP	XQ1XQ	XR1TR	XS0AA

XT0CB	XT1XT	XU2TQ	XV1XV	XW1OC	XX1ON	XY0XY
XZ1AT	XZ9ER	ZR0ZR	Z5OTP			

The following is a list of other macros (such as EQU) that deviate from the naming standard of starting with the letter I. For more information about these macro naming standards, see Table 1 on page 16.

BRPEQ	CAIEQ	CCLEQ	CLAWC	CLHEQ	CLKEQ	CPSEQ
CVHDK	CVRTK	CVTEQ	CZOCP	CZ1CP	CZ1SE	CZ3CP
CZ4CP	DADFQ	DATAS	DSEQU	ECBEQ	FIDEQ	GLBEQ
GLOB	GL0EQ	LINEQ	LOCEQ	MAPEQ	MP0EQU	MRLNQ
NODEQ	OMTEQ	PARSE	PIUEQ	PTVEQ	QWEQU	REGACP
REGEQ	REGEQ1	REGSTR	REGVAL	RGEQUC	RITEQ	RMXEQ
RTCEQ	RTTEQ	SETX	SNAEQ	SSCPP	TAPEQ	PFGLB
TA0EQ	TRMEQ	TSTEQ	UCNFEQ	UMSG	UXTEQ	XMSEQ

The following is a list of TPFDF structured programming macros (SPMs) that deviate from the naming standards for TPF macros. For more information about macro naming standards, see Table 1 on page 16.

#	#ANALOP	#BOOLTRN	#CASE	#CAST	#CHKSTAC	#CONB
#COND	#CONH	#CONP	#CONS	#CONT	#CONX	#DECODEB
#DO	#DOEX	#DOPROC#	#DOSTAK#	#ECAS	#EDO	#EIF
#EIFM	#ELIF	#ELOP	#ELSE	#ESUB	#EXEC	#EXIF
#GETBC##	#GOTO	#IF	#IFPROC#	#LBIT	#LEVL	#LOCA
#OREL	#PERF	#POPAAC#	#POPINS#	#POPMAC#	#POPNEST	#PUSHAAC
#PUSHINS	#PUSHLAB	#PUSHMAC	#PUSHNES	#REGR###	#RPRT	#SPM
#SPRT	#STKINS#	#STPC	#STPF	#STPH	#STPR	#SUBR
#UBACK	#UENTR	#UEXIT	#URTRN			

TPF 4.1 Headers

ADATA	BLDTOL	C$AM0SG	C$BK0RP	C$BL0RP
C$CINFC	C$CJ6KP	C$CK1KE	C$CK2SN	C$CX0CK
C$DADFQ	C$DBSAC	C$DBSDC	C$DCTIST	C$EB0EB
C$GLOBZ	C$GW01SR	C$IBMHDR	C$ICADAP	C$ICCACB
C$ICHUTL	C$ICILI	C$ICOLR	C$ICPATH	C$ICRACB
C$ICUSDA	C$IC0CK	C$IDCUTL	C$IDDMEQ	C$IDIRMC
C$IDSALO	C$IDSCDR	C$IDSCRP	C$IDSEAT	C$IDSECR
C$IDSELD	C$IDSELT	C$IDSELV	C$IDSEMR	C$IDSEPD
C$IDSERD	C$IDSFRP	C$IDSICD	C$IDSICR	C$IDSIHR
C$IDSINQ	C$IDSIPL	C$IDSKPT	C$IDSLDR	C$IDSLDT
C$IDSLSD	C$IDSLST	C$IDSMXP	C$IDSOLD	C$IDSPAT
C$IDSPRG	C$IDSPVR	C$IDSSAL	C$IDSUXT	C$IDSXTP
C$IEDCTL	C$IFDOMC	C$IFLDDF	C$IHCTCB	C$IMERMC
C$IPTBL	C$IRDICB	C$ISCCDT	C$ISCFDT	C$ISCIPT
C$ISCLNT	C$ISDDCB	C$ISIUCV	C$ISMTCB	C$ISQLMC
C$ISRTBK	C$ISTPCB	C$ITPICB	C$ITRTBL	C$ITUUTL
C$KPTPAT	C$MI0MI	C$MK0CK	C$MS0AT	C$MS0UT
C$PI1DT	C$RC0PL	C$RECOUP	C$RMXEQ	C$RV1VT
C$SD0RV	C$SI3CT	C$SQLCA	C$SQLDA	C$SRCK1P
C$SRHH1P	C$STDHD	C$SYSEQ	C$SYSTC	C$SYSUG
C$TPLDR	C$UATBC	C$UCNFEQ	C$UD0RV	CLAW

COIBM	CTOOL	GLDEF	GLDEFH	LIBIH
MPHDR	MPSIPCC	NLDTCI	NLDTIF	PIUTR
PLFAPI	SYSAPI	TPFAPI	TPFARAPI	TPFCTYPE
TPFEQ	TPFERRNO	TPFFLOAT	TPFGLBL	TPFIO
TPFLIMIT	TPFLINK	TPFLOCAL	TPFMAP	TPFMATH
TPFMCSLL	TPFPARFT	TPFPARSE	TPFREGS	TPFSTARG
TPFSTDEF	TPFSTDIO	TPFSTDLB	TPFSTRNG	TPFTAPE
TPFTIME	TPFTUU	TPPC	ZIMAG	ZIMAGEQ
ZIMAGMC	ZIMAGMSG	ZTPLD		

Appendix B. Existing Database Record ID Deviations

The following is a list of database record IDs that have been identified by IBM as in-use on TPF systems and deviate from the range of X'0000' to X'00FF' and X'FC00' to X'FFFF'. See *TPF System Generation* for a list of record IDs that are verified during system initialization by the RAMFIL macro and for a list of required record IDs.

X'8384' (C'ca')
X'C1C1' (C'AA')
X'C1D6' (C'AO')
X'C1D9' (C'AR')
X'C2D2' (C'BK')
X'C2D3' (C'BL')
X'C2E4' (C'BU')
X'C3C1' (C'CA')
X'C3C2' (C'CB')
X'C3C3' (C'CC')
X'C3C4' (C'CD')
X'C3C6' (C'CF')
X'C3C7' (C'CG')
X'C3C9' (C'CI')
X'C3D1' (C'CJ')
X'C3D2' (C'CK')
X'C3D3' (C'CL')
X'C3D4' (C'CM')
X'C3D6' (C'CO')
X'C3D7' (C'CP')
X'C3D9' (C'CR')
X'C3E2' (C'CS')
X'C3E3' (C'CT')
X'C3E4' (C'CU')
X'C3E5' (C'CV')
X'C3E7' (C'CX')
X'C3E8' (C'CY')
X'C3F1' (C'C1')
X'C3F8' (C'C8')
X'C3F9' (C'C9')
X'C4C1' (C'DA')
X'C4C2' (C'DB')
X'C4C3' (C'DC')
X'C4C4' (C'DD')
X'C4D3' (C'DL')
X'C4D9' (C'DR')
X'C4E2' (C'DS')
X'C4E4' (C'DU')
X'C4E7' (C'DX')
X'C4F4' (C'D4')
X'C5C5' (C'EE')
X'C5D7' (C'EP')
X'C5D9' (C'ER')
X'C5E2' (C'ES')
X'C5F5' (C'E5')
X'C5F6' (C'E6')
X'C5F7' (C'E7')
X'C6C3' (C'FC')
X'C6C4' (C'FD')
X'C6C5' (C'FE')
X'C6C6' (C'FF')
X'C6C9' (C'FI')
X'C6D2' (C'FK')
X'C6D8' (C'FQ')
X'C6D9' (C'FR')
X'C6E8' (C'FY')
X'C7C5' (C'GE')
X'C7D3' (C'GL')
X'C7D6' (C'GO')
X'C7E2' (C'GS')
X'C8D4' (C'HM')
X'C9C3' (C'IC')
X'C9C4' (C'ID')
X'C9C6' (C'IF')
X'C9C8' (C'IH')
X'C9D4' (C'IM')
X'C9D9' (C'IR')
X'D2C5' (C'KE')
X'D2D7' (C'KP')
X'D2E3' (C'KT')
X'D2E5' (C'KV')
X'D2E6' (C'KW')
X'D3C3' (C'LC')
X'D3C4' (C'LD')
X'D3C8' (C'LH')
X'D3D3' (C'LL')
X'D3E2' (C'LS')
X'D3E7' (C'LX')
X'D4C1' (C'MA')
X'D4C2' (C'MB')
X'D4C3' (C'MC')
X'D4C4' (C'MD')
X'D4C6' (C'MF')
X'D4C9' (C'MI')
X'D4D3' (C'ML')
X'D4D6' (C'MO')
X'D4D7' (C'MP')
X'D4E2' (C'MS')
X'D4E3' (C'MT')
X'D4E7' (C'MX')
X'D4F4' (C'M4')
X'D5C1' (C'NA')
X'D5C2' (C'NB')
X'D5C3' (C'NC')
X'D5C4' (C'ND')
X'D5C6' (C'NF')
X'D5E2' (C'NS')
X'D6C1' (C'OA')
X'D6C6' (C'OF')
X'D6D3' (C'OL')
X'D6D4' (C'OM')
X'D6D7' (C'OP')
X'D6D9' (C'OR')
X'D6E3' (C'OT')
X'D6E7' (C'OX')
X'D7C1' (C'PA')
X'D7C4' (C'PD')
X'D7C7' (C'PG')
X'D7D2' (C'PK')
X'D7D3' (C'PL')
X'D7D4' (C'PM')
X'D7D6' (C'PO')
X'D7D7' (C'PP')
X'D7D9' (C'PR')
X'D7E2' (C'PS')
X'D7E6' (C'PW')
X'D7F1' (C'P1')
X'D7F2' (C'P2')
X'D7F3' (C'P3')
X'D7F4' (C'P4')
X'D7F5' (C'P5')
X'D7F6' (C'P6')
X'D8D9' (C'QR')
X'D9C3' (C'RC')
X'D9D3' (C'RL')
X'D9D4' (C'RM')
X'D9D7' (C'RP')
X'D9E3' (C'RT')
X'D9E7' (C'RX')
X'E2C1' (C'SA')
X'E2C4' (C'SD')
X'E2C6' (C'SF')
X'E2C9' (C'SI')
X'E2D2' (C'SK')
X'E2D3' (C'SL')
X'E2D6' (C'SO')
X'E2D7' (C'SP')
X'E2D8' (C'SQ')

X'E2D9' (C’SR’)
X'E2F1' (C’S1’)
X'E2F6' (C’S6’)
X'E25C'
X'E3C5' (C’TE’)
X'E3D3' (C’TL’)
X'E3E2' (C’TS’)
X'E4C2' (C’UB’)
X'E4D4' (C’UM’)
X'E4E2' (C’US’)
X'E4E5' (C’UV’)
X'E4E9' (C’UZ’)
X'E5D9' (C’VR’)
X'E7C2' (C’XB’)
X'E7C5' (C’XE’)
X'E7C9' (C’XI’)
X'E7D3' (C’XL’)
X'E7D7' (C’XP’)
X'E7E2' (C’XS’)
X'E7E5' (C’XV’)
X'E7E8' (C’XY’)
X'E9D9' (C’ZR’)
X'E2E2' (C’SS’)
X'E2F2' (C’S2’)
X'E2F7' (C’S7’)
X'E3C1' (C’TA’)
X'E3C6' (C’TF’)
X'E3D4' (C’TM’)
X'E3E9' (C’TZ’)
X'E4C4' (C’UD’)
X'E4D8' (C’UQ’)
X'E4E3' (C’UT’)
X'E4E6' (C’UW’)
X'E5C1' (C’VA’)
X'E5E4' (C’VU’)
X'E7C3' (C’XC’)
X'E7C6' (C’XF’)
X'E7D1' (C’XJ’)
X'E7D4' (C’XM’)
X'E7D8' (C’XQ’)
X'E7E3' (C’XT’)
X'E7E6' (C’XW’)
X'E7E9' (C’XZ’)
X'E9E2' (C’ZS’)
X'E2E3' (C’ST’)
X'E2F3' (C’S3’)
X'E2F8' (C’S8’)
X'E3C3' (C’TC’)
X'E3D2' (C’TK’)
X'E3D9' (C’TR’)
X'E4C1' (C’UA’)
X'E4C6' (C’UF’)
X'E4D9' (C’UR’)
X'E4E4' (C’UU’)
X'E4E8' (C’UY’)
X'E5C4' (C’VD’)
X'E7C1' (C’XA’)
X'E7C4' (C’XD’)
X'E7C8' (C’XH’)
X'E7D2' (C’XK’)
X'E7D5' (C’XN’)
X'E7D9' (C’XR’)
X'E7E4' (C’XU’)
X'E7E7' (C’XX’)
X'E9D3' (C’ZL’)

Appendix C. Record IDs Used for RTA/RTL Tape

The following record IDs are used when logging records to the RTA/RTL tape.

Table 3. Record IDs for RTA/RTL

Record ID	Description
"BM"	Inboard block multiplexor code
"CA"	Long term pool return record
"CB"	DASD dispensing log record
"CT"	CCP trace record
"CW"	CRW logout record
"DB"	CUAK error ID
"DD"	Dump separator record
"DE"	Device error log message ID
"DF"	Valid dump record
"DG"	Recoup record
"DP"	Dump data record
"DX"	Primary Dump Label index record
"DY"	Secondary Dump Label index record
"MD"	3705 MDR code
"MR"	3270 BSC MDR input record
"MX"	Inboard multiplexor code
"OB"	3270 Local/3215/3705 OBR input record
"RT"	Real-time trace record
"SH"	Subchannel logout record
"SD"	Selective dump record
"SL"	Inboard selector channel code
"ST"	Selective trace record
"SY"	Sysplex Timer (STR) record
"TD"	Reserved for IBM use
"TR"	PIU trace record
"TV"	Test Vehicle record
"UC"	Unknown channel type code
X'00E3'	MPIF MSRB record
X'00E4'	SNAP Dump record
X'00E5'	Online mini dump
X'33xx'	EREP DASD record
X'3480'	MDR record for 3480/3490
X'3420'	OBR Tape error log record

Index

P

R

S

T

U

V

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.

SH31-0165-10