Airline Control System Version 2

Application Programming Reference – C Language

Release 4.1

SH19-6950-14

Airline Control System Version 2

Application Programming Reference – C Language

Release 4.1

SH19-6950-14

Note!

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

Fifteenth Edition (April 2011)

This edition applies to Release 4, Modification Level 1, of Airline Control System Version 2, Program Number 5695-068, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

ALCS Development
2455 South Road
P923
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

Figures

Notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights 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:

The Director of Licencing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
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
522 South Road
Mail Drop P131
Poughkeepsie, NY 12601-5400
USA

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

Programming interface information

This Application Programming Reference is intended to help programmers write applications programs to run under Airline Control System Version 2, Program Number 5695-068. This Application Programming Reference documents General-Use Programming Interface and Associated Guidance Information provided by Airline Control System Version 2 Program Number 5695-068.

General-Use programming interfaces allow the customer to write programs that obtain the services of Airline Control System Version 2.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

About this book

This book provides reference material for programmers writing application programs in C language for Airline Control System Version 2 Release 4 Modification Level 1 (ALCS).

ALCS is one of a family of IBM* programs designed to satisfy the needs of airlines and other industries with similar requirements for high-volume and high-availability transaction processing.

The product, which is also known as TPF/MVS, provides the Transaction Processing Facility (TPF) application programming interface (API) for z/OS* environments. It supersedes ALCS/Multiple Virtual Storage/Extended Architecture (ALCS/MVS/XA*), known as ALCS Version 1.

Throughout this book:

- Airline Control System Version 2 is abbreviated to ALCS unless the context makes it necessary to distinguish between ALCS Version 2 Release 4.1, and the predecessor products.
- Airlines Line Control Interconnection (ALCI) includes the function of network extension facility (NEF).
- Advanced Communications Function for the Virtual Telecommunication Method is abbreviated to VTAM*.
- TPF refers to all versions of Transaction Processing Facility and its predecessor, Airlines Control Program (ACP).
- MVS* refers to z/OS.

Who should read this book

This book is written for experienced C-language application programmers who have some familiarity with ALCS. As a minimum, you should have read the *ALCS Application Programming Guide*. Additional information on ALCS is available in the publications listed in the “Bibliography” on page 269.

Although the book is primarily intended for application programmers, some sections will also be of interest to system programmers.

How this book is organized

This book is organized as follows:

Chapter 1, “ALCS C programming conventions and facilities”
This chapter describes the special conventions and rules that you must follow when writing C programs for ALCS. It also lists the contents of the various header files that you can use with ALCS C programs.

Chapter 2, “ALCS API functions — reference”
This is the reference chapter. It includes descriptions of all of the C library functions provided specifically for the ALCS environment. The functions are arranged alphabetically.

Appendix A, "C library functions available under ISO-C, ALCS, and TPF"
This appendix lists the C library functions that you can use in ALCS programs.

Appendix B, "Acronyms and abbreviations"
This appendix contains a list of acronyms and abbreviations.

The book also includes a bibliography, a glossary of terms, and an index.

Chapter 1. ALCS C programming conventions and facilities

When you write C programs intended to run under ALCS, you must follow some conventions that are different from those that apply in other C programs. This chapter describes facilities that are available, and conventions that you should use, when programming ALCS applications in C. It also describes the header files supplied with ALCS.

1.1 C language compatibility

ALCS supports both the ISO-C language and TPF-C. Applications can be ported between ALCS and TPF, or between ALCS and other C platforms. For more details of compatibility see Appendix A, “C library functions available under ISO-C, ALCS, and TPF” on page 233.

To use ALCS application programs written in C, you must compile them with the IBM z/OS XL C/C++ compiler. To use DECBs in ALCS application programs written in C, you must compile them with the IBM z/OS XL C/C++ compiler. The *ALCS Application Programming Guide* describes how to compile and link-edit C programs.

1.2 Structure of C programs

This section describes the structure of C programs written for ALCS.

1.2.1 Source module

Each C language source module written for ALCS contains one or more of the following:

- `#include` statements; the C compiler treats the included files as part of the source program
- Functions defined in this module
- Calls to C functions in this source module
- Calls to C functions in other modules
- Calls to assembler modules
- Calls to other programs using the using `entdc` and `entrc` functions.

1.2.2 Load modules

Before you can use your C programs under ALCS, your system programmer must build one or more load modules which contain the programs. The *ALCS Application Programming Guide* describes how to create load modules from C source programs.

When the system programmer has built the load module or modules, they must be loaded onto the ALCS system. See the *ALCS Operation and Maintenance* for details.

1.3 Function names

You can use any names for functions that are allowed by the C compiler. When you call a function from the same source module, the C compiler uses the whole of the function name to identify which function you mean.

You can also call a C function defined in another program, provided the function you are calling is in a program in the same load module. See Figure 1.

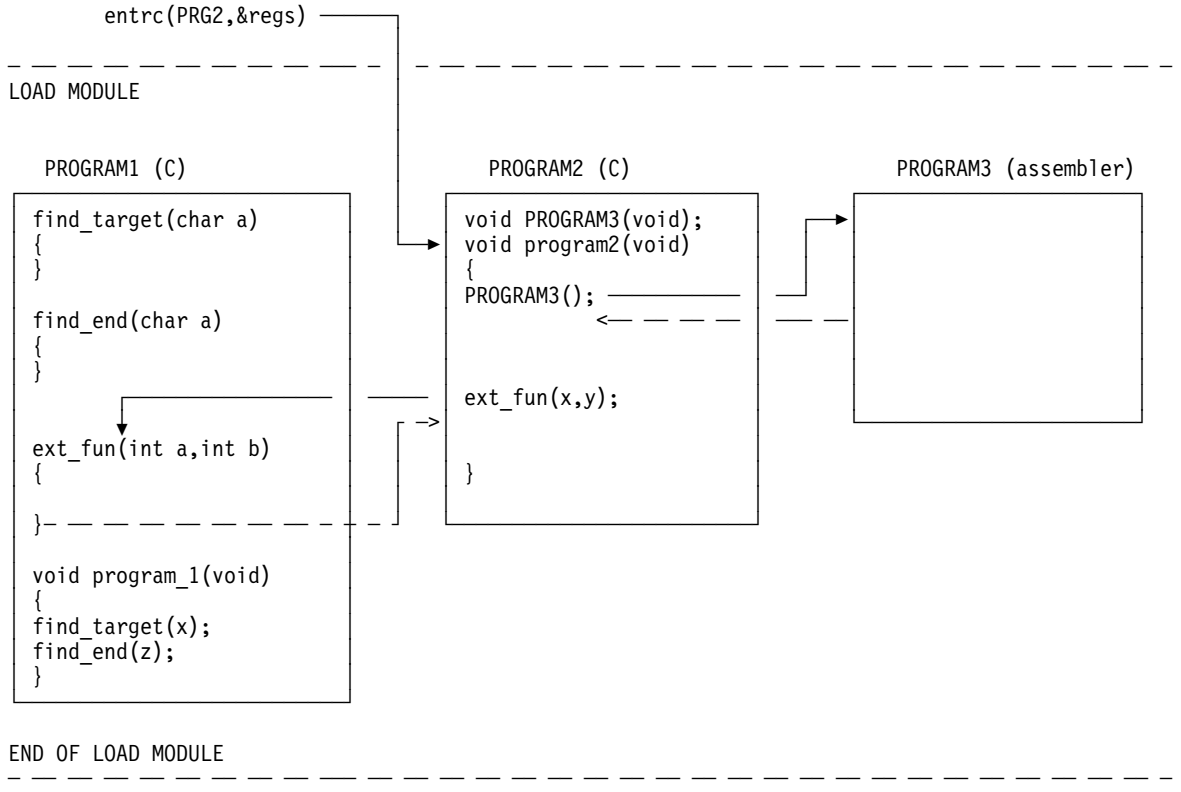


Figure 1. Calls between programs in the same load module

In Figure 1, 3 functions are shown defined in PROGRAM1.

`find_target`, `find_end`
: These two functions are only used internally in PROGRAM1.

`ext_fun`
: This is a function that is intended to be used by other C programs.

The C program called PROGRAM2 uses `ext_fun` and also makes a call to the assembler program PROGRAM3. The calls are shown by solid lines, the return paths from the called programs are shown in broken lines.

Note: Figure 1 also shows an `entrc` call into PROGRAM2 from a program outside this load module. This is the standard ALCS program linkage explained in the *ALCS Application Programming Guide*. (The 4-character name PRG2 has been associated with an entry point in program PROGRAM2 during the load module build process.)

1.3.1 Length of function names

The C language allows function names longer than the eight-character external names that the MVS linkage editor supports.

To resolve this, the z/OS Language Environment Prelinker (and the MVS linkage editor) or the binder allow you to map long names into short names for object modules produced by the compiler. You can also use the compiler directive `#pragma map`.

When referencing programs written in assembler, you can either use the short name of the program (8 characters or less) or map a suitable long name, using the `#pragma map` directive. For example, to call program PROGRAM3 by the name `calculate_top`, you could code the following:

```
#pragma map(calculate_top, "PROGRAM3")
```

1.4 Requirements for reentrant programs

All application programs intended to run under the control of ALCS must be reentrant. You must code each application program so that it can be used simultaneously by more than one entry.

It is possible (though difficult) to write self-modifying code in C. This is not allowed in ALCS C programs.

TPF compatibility

1. ALCS (but not TPF) supports data objects of storage class `extern`. If your programs must be compatible with TPF, do not declare data objects with storage class `extern static`, or with the keyword `extern`. Only use storage classes `automatic`, `register`, `const`, and `volatile`, and within the scope of a file or block, the storage class `static`.
2. The enter/back services of TPF may cause some program segments to be relocated in storage during the processing of an ECB. For compatibility with TPF, do not take the addresses of functions and do not use pointers to functions.
3. Application programs intended to run under TPF must not exceed 4KB in size. Application programs that are intended to run under ALCS can be any size.

There are currently two implementations of the C language for the TPF Version 4 Release 1 product. One implementation is called TARGET(TPF) and is specific to the TPF product because the IBM z/OS XL C/C++ compiler produces TPF-specific code. The second implementation is called ISO-C and supports the standard object code produced by the IBM z/OS XL C/C++ compiler. The ISO-C implementation supports all C language constructs. Both implementations require TPF header files and runtime libraries.

Note: These restrictions apply to TARGET(TPF) but not to ISO-C support in TPF.

1.4.1 Strings

The z/OS XL C/C++ compiler makes text strings writeable by default. This means that at runtime a copy of each string is placed in writeable storage.

You can use the `#pragma strings(readonly)` directive to place the strings into read-only storage. This is recommended for compatibility with TPF-C and to avoid the overheads of initializing the writeable strings.

Note: You must code the `#pragma strings(readonly)` directive before the `#include <tpfeq.h>` directive.

1.4.2 Header files

The ALCS header file `<tpfeq.h>` includes the system function prototypes and constants needed in all ALCS C programs. Include `<tpfeq.h>` as the first header file in every ALCS C program.

You might need to include additional header files, depending on what functions your program uses (see 1.9, “ALCS header files” on page 7). You might also need to include header files written specifically for your installation.

You can create your own header file which contains all these header files, together with `#pragma`s, `#define`s, and so on that are required in your application. For example, your header file might contain the following:

```
#pragma strings(readonly)
#include <tpfeq.h>
#include <tpfio.h>
#include <tpfapi.h>
#include <stdio.h>
#include <strings.h>
```

If this source file is named MYAPPHDR, you can include it in all your application programs by coding:

```
#include "myapphdr.h"
```

Recommendations

When creating header files, you are advised to follow the following rules:

- Always use the same structure to reference the same record. Put these structures in common header files that programmers can include when needed. In addition, all records should use standard headers.
- Do not use names starting with `dxc` or `tpf` for header files or programs. These file names are used by ALCS.
- Do not define preprocessor or C language variables starting with the characters: `DXC`, `dxc`, `TPF`, or `tpf`. Your names might conflict with names used by ALCS.
- ALCS uses file names starting with the characters `c$` for header files that have been created from assembler DSECT macro definitions. Do not use such names for other files.

  Note: The IBM z/OS XL C/C++ compiler provides a DSECT conversion utility which generates a structure to map an assembler DSECT. This utility is described in the *z/OS XL C/C++ User's Guide*. You might find this utility useful for creating structures for existing records in your installation.

- Avoid expanding the same header file more than once. One method that helps avoid this is to precede each `#define` in a header file with a `#ifndef`. For example:

```
#ifndef xxxxxx
#define xxxxxx
  ⋮
(text of the header file)
  ⋮
#endif
```

  In this case, if *xxxxxx* has been defined previously (possibly because this header file has already been included), the preprocessor does not include the header file text a second or subsequent time.

1.5 Defining structures

When you define structures in ALCS C programs (for example, to describe the layout of data in a record) you must consider boundary alignment.

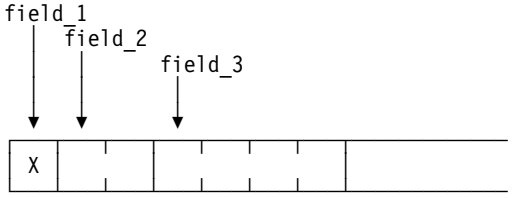
By default, the z/OS XL C/C++ compiler aligns field boundaries according to the type of each field defined in the structure (Figure 2).

Figure 2. Default alignments of fields in structures

Data Type	Storage occupied	Alignment
`char`	1 byte	byte
`int`	4 bytes	fullword
`short int`	2 bytes	halfword
`long int`	4 bytes	fullword
`float`	4 bytes	fullword
`double`	8 bytes	doubleword

When you define `int`, `short int`, and `long int` fields within a structure, the C compiler normally forces boundary alignments so that each new field starts at a fullword or halfword boundary.

Sometimes, you might need to override these default boundary alignments. For example, suppose you have to define the following fields in a structure:



Assume `field_2` is of type `short` and `field_3` is of type `long`. Both fields must follow `field_1` without any gaps caused by forced alignment.

You can do this by defining the structure as a `_Packed struct` as follows:

```
_Packed struct example
{
  char  field_1;
  short field_2;
  long  field_3;

  ⋮
};
```

You must also use the `_Packed` keyword any time that you define an instance of this structure. For example:

```
struct _Packed example ex1, *ptr_ex2;   /* define an instance of the structure */
                                        /* define a pointer to the structure   */
```

1.6 Time and date functions

All the C time and date functions are available. They use the current C locale and the MVS system clocks.

The `timec` function is provided to access the ALCS time and date services directly. (Remember that the ALCS system time and date may be different from the MVS time and date.)

TPF compatibility

Applications written for TPF that use the `time` function to get the TPF system time may need to be modified for use with ALCS. IBM recommends that you use the `timec` function to get the ALCS system time.

The `tpf_STCK` function is provided to access the S/390 time-of-day (TOD) clock value for situations requiring greater accuracy than to the nearest second.

1.7 Input message formats

In C programs you normally use the `stdin` functions to read messages from the originating terminal. This is the recommended method. However, you can directly examine an input message in AMSG or IMSG format. (You must know which format the message is in.) 1.10.1, “AMSG contents for an input message” on page 8 describes how ALCS formats an AMSG input message and 1.11.1, “IMSG contents for an input message” on page 9 describes how ALCS formats an IMSG input message.

To examine an input message in AMSG format you must include the header file `<c$am0sg.h>`. To examine an input message in IMSG format you must include the header file `<c$cm1cm.h>`.

1.8 Routing of messages

In C programs you normally use the `stdout` functions to send messages back to the originating terminal. However, you can also use the `routc` function to do this. In addition, `routc` allows you to send a message to another terminal or to an application. When you call `routc` you must supply it with the address of a **routing control parameter list** (RCPL).

1.8.1 The RCPL

When ALCS receives a message it creates an ECB and attaches the message on storage level 0. It also puts information relating to the origin of the message in an area in the ECB identified by the label `ce1rcpl` (see the *ALCS Application Programming Guide*). This area contains an RCPL with the format described in 1.16.1, “RCPL supplied with an input message” on page 15.

You can modify (or copy and modify) this RCPL to create a different RCPL which you can then use with a `routc` function (see 1.16.2, “RCPL required for an output message” on page 16). Alternatively, you can create an RCPL for an output message from scratch.

The header file <c$rc0pl.h> defines the contents of a RCPL. You must include this header file when you use `routc` in a program. (See 1.16, “<c$rc0pl.h>” on page 14.) When the output message is in AMSG format, you must also include the header file <c$am0sg.h>. When the output message is in OMSG format, you must also include the header file <c$cm1cm.h>. You must also set bit RCPL2REC in `rcplctl2` to 0 when the message is in AMSG format and to 1 when the message is in OMSG format. (See 1.16.2, “RCPL required for an output message” on page 16.)

1.9 ALCS header files

ALCS provides header files, some of which you must include in all ALCS application programs written in C, and some of which you only need to include when you are using certain functions. (See Chapter 2, “ALCS API functions — reference”.)

The header files perform the following functions:

- Map the ECB and other ALCS data structures
- Provide C function prototypes
- Define constants for the function parameters and return codes
- Define macros for the functions implemented as macros
- Define the `TPF_regs` structure (`TPF_regs` is a C structure used to pass parameter values in registers).

All ALCS programs require you to include the file <tpfeq.h>, which contains the definitions required by all ALCS C functions, as the first header file. Some functions require additional header files, for example, `attac` requires <tpfapi.h>. These header file requirements are shown in the function descriptions in Chapter 2, “ALCS API functions — reference”

1.10 <c$am0sg.h>

This header file defines the application message format (AMSG) structure. In ALCS, input and output messages in AMSG format are contained in one storage block.

You do not need to include <c$am0sg.h> when you are using the standard C input/output functions (for example, `gets`, `puts`, and `printf`). However, you must

include this header file when you are examining an input message in AMSG format or using the routc function to output a message in AMSG format.

1.10.1 AMSG contents for an input message

ALCS sets up the following fields in the storage block when it passes a message in AMSG format to a program.

am0rid
: This 2-byte char array contains the characters 'MI'.

am0cct
: This short int field contains the number of bytes in the message text, plus 5.

am0ccx
: This int field contains the number of bytes in the message text plus 5. am0ccx is used instead of am0cct when large messages in extended format are used.

am0txt
: This char array contains input text. Each line is terminated by the new-line character (_CAR). The message is terminated by an end-of-message character (_EOM or _EOU). The terminating character is included in the message length.

1.10.2 AMSG contents for an output message

Your application must set up the following fields in the storage block when it passes a message in AMSG format to ALCS:

am0rid
: Insert the characters 'OM' in this 2-byte char array.

am0cct
: Set this short int field to contain the number of bytes in the message text, including the end-of-message character, plus 5.

am0ccx
: Set this int field to contain the number of bytes in the message text plus 5. am0ccx is used instead of am0cct when large messages in extended format are used.

am0np1
: Set this char field to contain a _NOP character.

am0np2
: Set this char field to contain a _NOP character.

am0txt
: Put the message text into this char array. For details of the layout, see *ALCS Application Programming Guide*.

Example

The following example shows how to create an output message in AMSG format.

```
short int cct;
struct am0sg *am;

/* build an amsg format message for use with routc             */
am = getcc(D7,GETCC_TYPE,L1); /* get storage block for message */
cct = strlen(msg);            /* get length of message text    */
memcpy(am->am0txt,msg,cct);   /* move the text into the omsg   */
am->am0txt[cct++] = _EOM;     /* add _EOM and increment count  */
cct += 5;                     /* add length of fixed fields    */
am->am0cct=cct;               /* insert count into omsg        */
am->am0np1 = _NOP;
am->am0np2 = _NOP;
```

1.11 <c$cm1cm.h>

TPF compatibility

Do not use this header file in programs that must be compatible with TPF.

This header file defines the IMSG and OMSG structure. In ALCS, input messages in IMSG format and output messages in OMSGS format are contained in one storage block.

You do not need to include <c$cm1cm.h> when you are using the standard C input/output functions (for example, `gets`, `puts`, and `printf`). However, you must include this header file when you are examining an input message in IMSG format or using the `routc` function to output a message in OMSG format.

1.11.1 IMSG contents for an input message

ALCS sets up the following fields in a storage block when it passes a message in IMSG format to a program:

`cm1bid`
: This 2-byte `unsigned char` array contains the characters `'IM'`.

`cm1cct`
: This `short int` field contains the number of bytes in the message text, including the end-of-message character, plus 3.

`cm1ccx`
: This `int` field contains the number of bytes in the message text plus 3. `cm1ccx` is used instead of `cm1cct` when large messages in extended format are used.

`cm1cri`
: This 3-byte `unsigned char` array contains the communication resource identifier (CRI) of the originating terminal.

`cm1txi`
: This `char` array contains input text. Each line, except the last, is terminated by the new-line character (_CAR). The message is terminated by an end-of-message character (_EOM or _EOU).

1.11.2 OMSG contents for an output message

Your application must set up the following fields in 1 or more storage blocks when it passes a message in OMSG format to ALCS:

`cm1bid`
Insert the characters `'OM'` in this 2 `char` array.

`cm1cct`
Set this `short int` field to contain the number of bytes in the message text, plus 5.

`cm1ccx`
Set this `int` field to contain the number of bytes in the message text plus 5. `cm1ccx` is used instead of `cm1cct` when large messages in extended format are used.

`cm1cmw`
Set this `char` field to contain a _NOP character.

`cm1lna`
This `unsigned char` field optionally contains a display line number (ALCS outputs the message starting at this line on the display). When the destination is a display, set this field to a _NOP or _CAR character, or to an actual display line number plus hexadecimal 80 (0x80). (Line numbers start at zero.) When the destination is a printer, set this field to a _CAR character.

`cm1txt`
Put the message text into this `char` array. Terminate each line, except the last, with a new-line character (_CAR). For messages to terminals, terminate the message with the 3 character sequence: _CAR, _SOM, _EOM. For messages to printers, terminate the message with the 2 characters _CAR, _EOM.

Example

The following example shows how to create an output message in OMSG format.

```
char *msg = "message text 1\nmessage text 2";
short int cct;
struct cm1cm *cm;

/* build an omsg format message for use with routc             */
cm = getcc(D7,GETCC_TYPE,L1); /* get storage block for message */
cct = strlen(msg);            /* get length of message text    */
memcpy(cm->cm1txt,msg,cct);   /* move the text into the omsg   */
cm->cm1txt[cct++] = _EOM;     /* add _EOM and increment count  */
cct += 5;                     /* add length of fixed fields    */
cm->cm1cc1=cct;               /* insert count into omsg        */
cm->cm1lna = 0;               /* set line number 0             */
cm->cm1cmw = _NOP;
```

1.12 <c$co0ic.h>

TPF compatibility

Do not use fields defined in this header file in programs that must be compatible with TPF.

This header file defines the co0ic structure used with the function comic. You do not need to include this header file in programs, it is included by <tpfapi.h>. The comic function returns values in the following fields:

icecrn
: This 8-byte char array contains the communication resource name (CRN) of the resource.

icecri
: This unsigned int field contains the CRI of the resource.

iceldi
: This 8-bit unsigned int field contains the LDI of the resource. (This is the part of the CRI.)

icealc
: This unsigned int field contains one of the following:

  - CRI of the virtual SLC link associated with this X.25 permanent virtual circuit (PVC) (if any).
  - CRI of the virtual SLC link associated with this TCP/IP resource (if any).
  - CRI of the X.25 PVC or TCP/IP resource associated with this virtual SLC link (if any).

icenlc
: This unsigned int field contains the CRI of the next LU 6.1 parallel session in a chain of parallel sessions. The chain is anchored in the communication table entry for the controlling LU 6.1 link for these parallel sessions.

icensc
: This unsigned int field contains the CRI of the next TCP/IP dynamic server in a chain of TCP/IP resources. The chain is anchored in the communication table entry for their controlling TCP/IP server resource.

icearc
: This unsigned int field contains the CRI of the associated resource (if any).

iceorn
: This unsigned int field contains the communication resource ordinal number (a unique number associated with the communication resource).

iceusl
: This long int field contains the length of user data area (if any) in the communication tables.

icesze
: This 8-bit unsigned int field contains the number of display lines (rows) when the resource is a display terminal.

iceaty
: This 8-bit unsigned int field contains the device type.

iceaci
: This 8-bit unsigned int field contains the alternate CRAS index (AT number) or alternate CRAS printer index (AP number).

icepcl
: This 8-bit unsigned int field contains one of the following:

  - SLC link protocol

- X.25 PVC type
- APPC connection type
- Application protocol for a TCP/IP connection.

`icearn`
This 8-byte `char` array contains the application name when the resource is a terminal, and when it is routed to an application. The application name is left justified and the array is padded with spaces.

This name is not necessarily the same as the application program name of the input message editor program for the application.

`iceprg`
This 4-byte `char` array contains the application program name. When the resource is an application, this array contains the application program name of the input message editor program for the application.

`icebsz`
This 2-byte unsigned int field contains the printer buffer size (number of bytes). For example, if the resource is a terminal with a 1920 character buffer, this field contains X'0780' (decimal 1920).

`iceoid`
This 8-bit unsigned int field contains the communication ID (COMID) of the system that owns the resource.

`icepid`
This 8-bit unsigned int field contains the communication ID (COMID) of the system that the resource connects to.

`iceorc`
This unsigned int field contains one of the following:

- CRI of owning ALCI LU for an ALC terminal connected through ALCI.
- CRI of owning NEF2 LU for an ALC terminal connected through NEF2.
- CRI of owning SLC link for an ALC terminal connected through SLC.
- CRI of owning X.25 PVC for an ALC terminal connected through X.25.
- CRI of owning TCP/IP server connection for an ALCterminal connected through TCP/IP.
- CRI of owning MQ resource for an ALC or 3270-like terminal connected through the MQ Bridge.
- CRI of owning WAS resource for an ALC or 3270-like terminal connected through the WAS Bridge.
- CRI of owning TCP/IP server connection for a TCP/IP dynamic server connection.
- CRI of owning LU 6.1 link for a parallel session within an LU 6.1 link.

`icecol`
This 8-bit unsigned int field contains the number of display columns (when the resource is a VTAM 3270 display terminal).

`iceipl`
This 2-byte unsigned int field contains the TCP/IP local port number.

`iceipr`
This 2-byte unsigned int field contains the TCP/IP remote port number.

`iceiph`
This 16-byte char array contains the TCP/IP remote IP address in dotted decimal notation, left-justified and padded with spaces.

This header file also defines the following bit fields that you can test. The meaning when each of these fields is set is shown below:

`ices3ex` Resource is a terminal that supports the IBM 3270 extended data stream.

`icesaaa` Terminal hold (also called AAA hold) is set.

`icesalc` Resource is an ALC terminal.

`icesape` Resource is an application that is permanently active. That is, the operator cannot make the application inactive.

`icesapn` Resource is an alternate CRAS printer.

`icesatn` Resource is a terminal with alternate CRAS authority.

`icesbbb` IBM 3270 terminal is in the SNA in-bracket state (that is, if ALCS has received a message with SNA begin bracket from the terminal).

`icescmd` Resource is an application that can process ALCS commands (the application processes input messages with primary action code Z).

`icescst` Resource is in session with ALCS (active).

`icesdbc` Resource is a terminal that supports the IBM 3270 double-byte character set.

`icesfal` esource has Alternate CRAS authority but is not one of the unique AT1 through AT255 CRAS devices.

`icesfpr` Resource has Prime CRAS authority but is not the unique Prime CRAS device.

`icesipc` Resource is a TCP/IP client connection.

`icesipd` Resource is a TCP/IP dynamic server connection.

`icesips` Resource is a TCP/IP server connection.

`icesiss` System generated messages are not sent to this terminal resource.

`iceslog` Resource is a terminal and is routed to an application.

`icesmqt` Resource is an ALC or 3270-like terminal attached through the MQ Bridge.

`icesnef` Resource is an ALC terminal, attached through an ALCI link, or an ALC terminal attached through NEF2.

`icesprc` Resource is a terminal with Prime CRAS authority.

`icesroc` Resource is the read only (`RO`) CRAS terminal.

`icesslc` Resource is an ALC terminal attached through an SLC link.

`icestcp` Resource is controlled by this ALCS.

`icestpp` Resource is a printer.

`icestps` Resource is a display.

`icesuns` Resource is unusable.

`iceswst` Resource is an ALC or 3270-like terminal attached through the WAS Bridge.

`icesx25` Resource is an ALC terminal attached through an AX.25 link.

Example

The following example shows how you could test one of these bit fields:

```
struct co0ic *p;

if (p->icescst)
   printf("Resource is in session with ALCS\n");
```

1.13 <c$decb.h>

This header file defines (as type TPF_DECB) the structure of the ALCS data event control block (DECB). The DECB is described in the *ALCS Application Programming Guide*.

You must include this header file when using any of the DECB management functions `tpf_decb_create`, `tpf_decb_locate`, `tpf_decb_release`, `tpf_decb_swapblk`, and `tpf_decb_validate`.

1.14 <c$eb0eb.h>

This header file defines the structure of the ALCS entry control block (ECB). The ECB is described in the *ALCS Application Programming Guide*.

You do not need to include this header file in programs. It is included by `<tpfeq.h>`.

1.15 <c$globz.h>

This header file defines the global tag definitions and is required if you are using the global area. It is created by the system programmer in your installation and is not part of ALCS. See *ALCS Installation and Customization* for details of how to create this file.

This file is included by `<tpfglbl.h>`.

1.16 <c$rc0pl.h>

This header file defines the routing control parameter list (RCPL) structure (see 1.8, “Routing of messages” on page 6). The fields in the RCPL have different meanings for input and output messages.

You do not need to include `<c$rc0pl.h>` when you are using the standard C input/output functions (for example, `gets`, `puts`, and `printf`). However, you must include this header file when you are using the `routc` function.

1.16.1 RCPL supplied with an input message

ALCS sets up the fields in an RCPL in the ECB (in the array `ce1rcpl`) when it receives a message:

`rcpldes`
When the message is to an application program, this 4-byte `char` array contains an application name. When the message is to a terminal, the first 3 bytes of the field contain the CRI of the destination terminal.

`rcplorg`
When ALCS received the message from a program, this 4-byte `char` array contains an application name. When ALCS received the message from a terminal, it contains the CRI of the originating terminal in the first 3 bytes of the field.

`rcplctl0`
This `char` field contains two non-exclusive bit settings which you can test using the following defined symbols:

RCPL0OTY
If set, `rcplorg` contains the origin application name, if unset, `rcplorg` contains the origin CRI.

RCPL0FMT
If set, the RCPL is expanded format (16 bytes), if unset, the RCPL is basic format (12 bytes).

Example

The following example shows how you could test one of these values:

```
#include <tpfeq.h>
#include <c$rc0pl.h>
  ⋮
  struct rc0pl *ptr_rcpl;
  ⋮
  /* set the pointer to the rc0pl area in the ECB */
  ptr_rcpl = (struct rc0pl *) ecbptr()->ce1rcpl;

  /* display the contents of the rcplorg field in the ECB */

  printf("The rcplorg field contains the");

  if (ptr_rcpl->rcplctl0 & RCPL0OTY)
  {
    printf("origin application name: %4.4s\n",ptr_rcpl->rcplorg);
  }
  else
  {
    printf("origin CRI: %02X%02X%02X\n",
           ptr_rcpl->rcplorgl,
           ptr_rcpl->rcplorgi,
           ptr_rcpl->rcplorgt);
  }
```

`rcplctl2`
This `char` field contains three non-exclusive bit settings which you can test using the following defined symbols:

RCPL2REC
: If set, the message is recoverable; if unset, the message is not recoverable.

RCPL2POS
: If set, the message is possibly a duplicate; if unset, the message is not a duplicate.

RCPL2FMH
: If set, the message text includes a function management header (FMH), otherwise it is unset.

Expanded RCPLs contain extra fields, as follows:

`rcplgdd`
: This `unsigned char` field contains bit switches. The first 4 high-order bits (bits 0, 1, 2, and 3) are reserved for ALCS use; you can use the other bits in accordance with your installation standards.

`rcplctr`
: This `unsigned char` field contains the length of the general data area that follows.

`rcplgda`
: General data area. This field is provided for compatibility with TPF.

1.16.2 RCPL required for an output message

Set up an RCPL as described below before calling the `routc` function to send a message. Set all unused bytes in the RCPL to binary zeros, using `memset` as shown in the “Example” which follows.

`rcpldes`
: When you send a message to an application program, set this 4-byte `char` array to contain an application name. When you send a message to a terminal, set the first 3 bytes of the field to the CRI of the destination terminal.

  Note: You can “broadcast” the same message to a number of terminals by calling `routc` several times, resetting the CRI value before each call.

`rcplorg`
: Set this 4-byte `char` array to the name of an application or store a CRI in the first 3 bytes.

`rcplctl0`
: Set this `char` field to contain five non-exclusive bit settings. Use the following defined symbols:

  RCPL0DTY
  : Set this to 1 when `rcpldes` contains the destination application name; set to 0 when `rcpldes` contains the destination CRI.

  RCPL0EXT
  : Set this to 1 if the message is in extended format for large messages, set to 0 when the message is in standard format

  RCPL0MTY
  : Set this to 1 when the message is unsolicited (not a response); set to 0 when the message is a response.

RCPL0RET
Set this to 1 when the message is being returned as a reply to the originator, otherwise set to 0.

RCPL0FMT
Set this to 1 when the RCPL is expanded format (16 bytes); set to 0 when the RCPL is basic format (12 bytes).

Example

The following example shows how you could set two of these bits:

```
struct rc0pl rcpl;                  /* define an RCPL      */
  ⋮
memset(&rcpl, 0, sizeof(rcpl));     /* clear RCPL to zeros */
  ⋮
/* set the rcpl to indicate:                                */
/*  rcpldes contains the destination application name      */
/*  the message is unsolicited                             */
rcpl.rcplctl0 = RCPL0DTY | RCPL0MTY;

memcpy(rcpl.rcpldes,"APPL",4); /* set the application to "APPL" */
  ⋮
```

`rcplctl2`
Set this `char` field to contain 3 non-exclusive bit settings. Use the following defined symbols:

RCPL2REL
Set this to 1 when you want `routc` to release the storage block that contains the message; otherwise set to 0.

RCPL2MFT
Set this to 1 when the message is in OMSG format. Set it to 0. when the message is in AMSG format.

RCPL2FMH
Set this to 1 when the message text includes a function management header (FMH), otherwise set to 0.

Expanded RCPLs contain extra fields, as follows:

`rcplgdd`
This `unsigned char` field contains bit switches. The first 4 high-order bits (bits 0, 1, 2, and 3) are reserved for ALCS use. You can use any of the other bits in accordance with your installation standards.

`rcplctr`
Set this `unsigned char` field to the length of the general data area that follows.

`rcplgda`
General data area. This field is provided for compatibility with TPF.

1.17 <c$stdhd.h>

This header file defines the structure of the standard ALCS record header. The standard header contains the following fields:

`stdbid`
Record identifier (2 bytes)

`stdchk`
Record code check (1 byte)

`stdctl`
Record control byte (1 byte)

`stdpgm`
Last update program name (4 bytes)

`stdfch`
Forward chain record address (4 bytes)

`stdbch`
Backward chain record address (4 bytes).

The *ALCS Application Programming Guide* explains the use of these fields in more detail.

1.18 <c$std8.h>

Compatibility

ALCS supports this header file only for compatibility. It is used in TPFDF for compatibility with TPF. IBM recommends that you do not use this header file in ALCS.

This header file defines the structure of the TPF standard record header for 8-byte file addresses. The standard header contains the following fields:

`istd8_bid`
Record identifier (2 bytes)

`istd8_chk`
Record code check (1 byte)

`istd8_ctl`
Record control byte (1 byte)

`istd8_pgm`
Last update program name (4 bytes)

`istd8_user`
Reserved for TPFDF or users of non-TPFDF databases (4 bytes)

`istd8_ibm`
Reserved for TPFDF (4 bytes)

`istd8_fch`
Forward chain record address (8 bytes)

istd8_bch
Backward chain record address (8 bytes).

Note: Applications that include <c$std8.h> must be compiled with the C++ compiler.

1.19 <tpfapi.h>

This header file defines ALCS API function prototypes (except for I/O functions) and constants, described in Chapter 2, “ALCS API functions — reference” on page 25.

<tpfapi.h> also defines the stdhdr structure which contains the following fields that ALCS requires for the rlcha function:

stdbid
This 2-byte char array contains the record ID.

stdrec
This unsigned char field contains the record code check (RCC).

stdfad
This unsigned int field contains the file address.

It also contains the the ronic_info_area structure which contains the following information which ALCS returns on a ronic call:

roncls
This unsigned short field contains the record class.

ronnbr
This unsigned long field contains the number of records allocated.

ronbty
This unsigned short field contains the block type code (L1, L2,...).

ronbln
This unsigned short field contains the logical block length (in bytes).

rontyp
This unsigned short field contains the fixed-file record type number, or pool interval number, or general file number.

ronic sets the following bit fields when the record is:

ronvfod
In delayed-file mode

ronvfop
Permanently resident

ronvfot
In time-initiated file mode

ronvfof
In force-read mode

ronvfou
In update (read before write) mode

`ronvfol`
Using long-term pool stamp processing

`ronvfoh`
Using hyperspace backing

`ronindc`
In a configuration data set

`ronindlt`
A long-term pool record

`ronindst`
A short-term pool record

`ronindp`
A pool record

`ronindg`
In a general file or general dataset

`ronindr`
A database record.

For an example of the use of this area, see 2.64, "`ronic` — Return information about a DASD record or record type" on page 153.

1.20 <tpfeq.h>

Include <tpfeq.h> as the first header file in every ALCS C program.

This header file defines the ALCS general system function prototypes and the following constants:

_CRI_PRC	Prime CRAS CRI
_CRI_ROC	RO CRAS CRI
_EUA	3270 erase unprotected to address
_FF	Form feed
_FID	ALCS field identifier
_FS	Field separator
_FSC	ALCS figure shift
_GE	Graphic escape
_HT	Horizontal tab
_IC	3270 insert cursor
_IFS	Interchange file seperator
_IGS	Interchange group seperator
_INP	Inhibit presentation
_IR	Index return
_IRS	Interchange record seperator
_IT	Indent tab
_IUS	Interchange unit seperator
_LF	Line feed
_LFEED	ALCS line feed
_LSC	ALCS letter shift
_NAK	Negative acknowledge
_NBS	Numeric backspace
_NGBLS	Number of global areas

_NL	New line
_NOP	ALCS no operation
_NSP	Numeric space
_POC	Programmed operator command
_PP	Presentation position
_PT	3270 program tab
_RFF	Required form feed
_RNL	Required new line
_RPT	Repeat
_RSP	Required space
_SA	Set attribute
_SBA	3270 set buffer address
_SBS	Subscript
_SEG	ALCS segment ID
_SEL	Select
_SHY	Syllable hyphen
_SI	Shift in
_SO	Shift out
_SOH	Start of header
_SOM	ALCS start of message
_SOS	Start of significance
_SP	Space
_SPS	Superscript
_STX	Start of text
_SUB	Substitute
_SW	Switch
_SYN	Synchronous idle
_TRN	Transparent
_UBS	Unit backspace
_UME	UMSG end of message
_VCS	SCS vertical channel select
_VT	Vertical tab
_WCW	ALCS write continue at cursor
_WEW	ALCS write and erase
_WLA	ALCS write on specified line
_WUN	ALCS write unsolicited

This header file also defines the `TPF_regs` structure and the enumeration type `t_regs`.

The `TPF_regs` structure is as follows:

```
/*
**   Struct TPF_regs: used to pass parameters to assembler
**      programs in registers.
*/

struct  TPF_regs
  {
  long int    r0;
  long int    r1;
  long int    r2;
  long int    r3;
  long int    r4;
  long int    r5;
  long int    r6;
  long int    r7;
  };
```

The enumeration type `t_regs` is as follows:

```
enum t_regs {R0, R1, R2, R3, R4, R5, R6, R7};
```

1.21 <tpfglbl.h>

This header file defines global interface function prototypes and constants. You must include <tpfglbl.h> in programs that use either of the global functions (`glob` or `global`).

1.22 <tpfio.h>

This header file defines the ALCS I/O function prototypes and constants. It includes the function prototypes and parameter definitions for all the I/O functions (except the sequential file functions). You must include <tpfio.h> in programs that use any of the I/O functions (find and file functions).

<tpfio.h> also contains the structure defined as type TPF_FAC8 which describes the parameter block used by the `tpf_fac8c` function.

The following input fields must be set before calling the `tpf_fac8c` function.

`ifacord`
: This `unsigned long long` field must be set to the record ordinal number.

`ifacrec`
: This 8-byte char array must be set to the fixed-file record type symbol when `ifactyp` contains IFAC8FCS. It must be left justified and padded with space characters.

`ifacrnb`
: This `unsigned long` field must be set to the fixed-file record type value when `ifactyp` contains IFAC8FCE.

`ifactyp`
: This `unsigned char` field must be set to indicate whether the record type specified is a symbol (IFAC8FCS) or a value (IFAC8FCE).

The following fields are returned by the `tpf_fac8c` function:

ifacadr
This field (defined type TPF_FA8) is set to the 8-byte file address.

ifacmax
This unsigned long long field is set to the maximum ordinal number for the requested record type.

ifacret
This unsigned char field is set to the return code, which is one of the following defined values:

TPF_FAC8_NRM
Normal return.

TPF_FAC8_TYP
Call type invalid or record type invalid.

TPF_FAC8_ORD
Record ordinal invalid.

1.23 <tpftape.h>

This header file provides the function prototypes and parameter definitions for the functions that access sequential files. You must include <tpftape.h> in programs which use functions that access sequential files.

<tpftape.h> also contains the tpstat structure, which contains the following fields. (The tdspc function returns information in these fields.)

tpname
A 3-byte char array containing the symbolic name of the sequential file

devtypr
An 8-byte char array containing the sequential file name

volser
A 6-byte char array containing the volume serial number

dsname
A 44-byte char array containing the data set name.

<tpftape.h> also defines the following bit fields. When these are set they indicate the following:

standby
Sequential file is standby

reserved
Sequential file is reserved

closed
Sequential file is closed

realtime
Sequential file is realtime

log
Sequential file is a log file

`diagnostic`
Sequential file is a diagnostic file

`general`
Sequential file is a general file

`input`
Sequential file is an input file.

Chapter 2. ALCS API functions — reference

This chapter contains an alphabetical listing of all ALCS API functions. The following information is shown for each function:

Format
The function prototype, and a description of any parameters.

Description
What service the function provides.

Normal return
What is returned when the requested service has been performed.

Error return
What is returned when the requested service could not be performed.

Note: Specifying invalid function parameters results in a system error (dump) with exit. Invalid parameters do not give an error return.

Loss of control
Some functions cause ALCS to remove control from the entry. This section describes the circumstances under which this happens. (System error with exit is not included as “Loss of control”.)

Example
One or more code segments, showing example function calls.

Related information
Where to find additional information relevant to this function.

Note: You must include <tpfeq.h> as the first header file in any ALCS C application program.

2.1 Common parameters

Some parameters are common to several functions. These are described in detail here.

level
One of 16 possible values representing a valid data level or storage level. Use one of the defined values D*x*, where *x* represents the hexadecimal number of the level (0-F). The action that the function takes with the *level* parameter is described with the function description.

program_name
Pointer to a program, or transfer vector, name in an ECB-controlled program. A program name conventionally consists of one alphabetic character followed by 3 alphanumeric characters. All alphabetic characters must be in uppercase (CAPITALS).

GDS
ALCS ignores this parameter (provided for compatibility with TPF).

NOTAG
When ALCS writes the record it does not insert the program name into the record header.

Compatibility

In TPF Version 3.0, all the file functions (`filec`, ...) allowed optional GDS and NOTAG parameter. All the find functions (`findc`,...) allowed an optional GDS parameter.

In later versions of TPF, the basic file and find functions (for example `filec`) do not allow the optional parameters GDS and NOTAG. Instead there is an extended version of each function (for example, `filec_ext`) which has a mandatory parameter *ext_file*. The programmer sets bit settings (defined values) in *ext_file* to indicate if the GDS and NOTAG option is required.

ext_file
With extended file functions (`filec_ext`, ...), set this parameter to one or more of the following values. (Use the + operator to specify more than one value.)

FILE_GDS
ALCS ignores this parameter value.

FILE_NOTAG
When ALCS writes the record it does not insert the program name into the record header.

FILE_DEFEXT
ALCS ignores this parameter value. Do not use with either FILE_NOTAG or FILE_GDS.

ext_find
With extended versions of find functions (`findc_ext`, ...), set this parameter to one of the following values:

FIND_GDS
ALCS ignores this parameter value.

FIND_DEFEXT
ALCS ignores this parameter value.

2.2 atawait — Wait for one or more asynchronous APPC/MVS calls to complete

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void atawait(void);
```

Description

Use the atawait function to wait for one or more outstanding asynchronous APPC/MVS calls to complete.

The atawait function waits only for asynchronous APPC calls to complete and does not wait for the completion of any other ALCS I/O operations. The I/O counter and error indicators in the ALCS ECB are not affected.

Note: APPC/MVS function calls can specify either synchronous or asynchronous notification of completion. This is determined by the Notify_type parameter (see *MVS Programming: Writing Transaction Programs for APPC/MVS*).

Normal return

Void. At least one of the asynchronous APPC/MVS functions issued by this ECB has completed. You must determine the status of each outstanding asynchronous APPC/MVS function by testing the post bit in the first byte of the event control block. You can determine the status of each function by testing the completion code in the event control block supplied with the Notify_type parameter.

Error Return

Not Applicable.

Loss of Control

This function can cause the entry to lose control.

Example

The following example shows a program calling the APPC/MVS send function on two conversations. It calls the send function asynchonously and uses atawait to test successful completion.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <atbpbc.h>          /* APPC/MVS C definitions         */


/* define fields used in APPC/MVS calls */

char * conv_id[2];           /* pointers to Conversation IDs     */
                             /* set by previous calls to ATBALLC */

long int ret_code[2] = {0,0};
long int send_and_flush = ATB_SEND_AND_FLUSH;
long int msg_length = 0;
long int none = ATB_NONE;
char msg[] = "test message .....................";
long int req_send_received[2] = {0,0};

/* define the notify parameters for asynchronous calls          */

struct  {
  long int indicator;        /* set to 1 for asynchronous call  */
  long int * evcb;           /* pointer to event control block  */
} notify[2];

long int appc_evcb[2];       /* event control blocks            */

/* define masks for the event control block                     */

#define POSTED    0x40000000 /* posted bit                      */
#define COMP_CODE 0x00FFFFFF /* completion code                 */
/*--------------------------------------------------------------*/
/* call the APPC/MVS send function asynchonously to send the    */
/* same message on two APPC conversations                       */
/*--------------------------------------------------------------*/

appc_evcb[0] =               /* initialize event control blocks */
appc_evcb[1] = 0;
notify[0].indicator =        /* indicate asynchronous request   */
notify[1].indicator = ATB_NOTIFY_TYPE_ECB;
notify[0].evcb = &appc_evcb[0]; /* set pointer to first evcb    */
notify[1].evcb = &appc_evcb[1]; /* set pointer to second evcb   */
msg_length = strlen(msg);    /* get the length of the message   */

ATBSEND(                     /* send call 1                     */
  conv_id[0],                /* In  - Conversation ID           */
  &send_and_flush,           /* In  - Send Type                 */
  &msg_length,               /* In  - Send Length               */
  &none,                     /* In  - Buffer ALET               */
  msg,                       /* In  - Buffer                    */
  &req_send_received[0],     /* Out - Req To Send Received      */
  (char *) &notify[0],       /* In  - Notify type               */
  &ret_code[0]);             /* Out - Return Code               */

if (ret_code[0] != ATB_OK)
{
  printf("ATBSEND call 1 not accepted - return code=%d/n",ret_code[0]);
  exit(0);
}
```

```
ATBSEND(                    /* send call 2                    */
  conv_id[1],               /* In  - Conversation ID          */
  &send_and_flush,          /* In  - Send Type                */
  &msg_length,              /* In  - Send Length              */
  &none,                    /* In  - Buffer ALET              */
  msg,                      /* In  - Buffer                   */
  &req_send_received[1],    /* Out - Req To Send Received     */
  (char *) &notify[1],      /* In  - Notify type              */
  &ret_code[1]);            /* Out - Return Code              */

if (ret_code[1] != ATB_OK)
{
  printf("ATBSEND call 2 not accepted - return code=%d/n",ret_code[1]);
  exit(0);
}

printf("ATBSEND calls accepted/n");

/*
     ...
     ... other processing
     ...
 */

/*-------------------------------------------------------------*/
/* wait for ATBSENDs to complete and check the completion codes */
/*-------------------------------------------------------------*/

do  /* wait until event control blocks for ATBSENDs are posted  */
{
  atawait();
} while (!(appc_evcb[0] & POSTED) && !(appc_evcb[1] & POSTED));

printf("atawait completed/n");

if ( (appc_evcb[0] & COMP_CODE) == ATB_OK)
{
  printf("ATBSEND call 1 succeeded/n");
}
else
{
  printf("ATBSEND call 1 failed -  completion code=%d/n",
          appc_evcb[0] & COMP_CODE);
}

if ( (appc_evcb[1] & COMP_CODE) == ATB_OK)
{
  printf("ATBSEND call 2 succeeded/n");
}
else
{
  printf("ATBSEND call 2 failed -  completion code=%d/n",
          appc_evcb[1] & COMP_CODE);
}
```

Related Information

MVS Programming: Writing Transaction Programs for APPC/MVS.
ALCS Application Programming Guide.
2.58, “mqawait — Wait for one or more asynchronous MQI calls to complete” on page 140.

2.3 attac — Attach a previously detached storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void *attac(enum t_lvl level);
```

level
A storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

Description

Use the attac function to reattach to the ECB the last storage block that the entry detached from the level specified in the *level* parameter. (The entry must have previously called a detac function).

The level specified in the *level* parameter must not have a storage block already attached.

Normal return

Pointer to the start of the storage address of the newly attached block.

ALCS restores the specified storage and data level to the contents at the time the most recent detac function was called for that level.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example reattaches a previously detached record to level 6 (D6).

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <c$cm1cm.h>

struct cm1cm *cm;
  ⋮
cm = (struct cm1cm *)attac(D6);
```

Related information

2.46, “getcc — Get a storage block” on page 116.
2.16, “detac — Detach a storage block” on page 52.

2.4 attac_ext — Attach a previously detached storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void *attac_ext(enum t_lvl level, int ext);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void *attac_ext(TPF_DECB *decb, int ext);
```

level
An ECB storage level (D0, D1,...DF.). See 2.1, “Common parameters” on page 25

decb
A pointer to a data event control block (DECB).

ext
Indicate which block is to be attached. Use one of the following terms:

ATTAC_USER_DEFAULT
Reattach the block from the most recent `detac_ext` at the specified ECB level or for the specified DECB.

ATTAC_USER_ACPDB
Reattach the block detached by a previous `detac_ext`. The block file address must match the file address at the specified ECB level or in the specified DECB. Store the required file address in the ECB data level (`ce1fm`*n*) or DECB (`idecfa`) before issuing `attac_ext`.

Description

Use the `attac_ext` function to reattach a block detached by a previous `detac_ext`. Specify the appropriate value for the *ext* parameter to indicate which block is to be attached.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Pointer of type void representing the address of the start of the newly attached block.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example detaches a storage block from ECB level 4 (D4) using detac_ext. Another block is obtained on ECB level 4 and also detached using detac_ext. Then the first block is attached using attac_ext. Note that the file address is saved and then re-stored on the appropriate level before issuing the attac_ext.

```
#include <tpfeq.h>
#include <tpfapi.h>

double   farw_save1;
double   farw_save2;

getfc(D4,GETFC_PRIME,"OM",GETFC_BLOCK,GETFC_NOSERRC);
memcpy(&farw_save1,&ecbptr()->ebcfw4,sizeof(double));
detac_ext(D4, DETAC_USER_ACPDB);

getfc(D4,GETFC_PRIME,"ZJ",GETFC_BLOCK,GETFC_SERRC);
memcpy(&farw_save2,&ecbptr()->ebcfw4,sizeof(double));
detac_ext(D4, DETAC_USER_DEFAULT);

memcpy(&ecbptr()->ebcfw4,&farw_save1,sizeof(double));
attac_ext(D4, ATTAC_USER_ACPDB);
```

This example reattaches a previously detached storage block to a DECB.

```
#include <tpfeq.h>
#include <tpfapi.h>

struct cm1cm *cm;
TPF_DECB     *decb;
  ⋮
cm = (struct cm1cm *)attac_ext(decb, ATTAC_USER_ACPDB);
```

Related information

2.3, “attac — Attach a previously detached storage block” on page 30.
2.5, “attac_id — Attach a previously detached storage block” on page 33.
2.16, “detac — Detach a storage block” on page 52.
2.17, “detac_ext — Detach a storage block” on page 53.
2.18, “detac_id — Detach a storage block” on page 55.
2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.

2.5 attac_id — Attach a previously detached storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void *attac_id(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void *attac_id(TPF_DECB decb);
```

level
An ECB storage level (D0, D1,...DF.). See 2.1, “Common parameters” on page 25

decb
A pointer to a data event control block (DECB).

Description

Use the attac_id function to reattach a block detached by a previous detac_id. The block file address must match the file address at the specified ECB level or in the specified DECB. Store the required file address in the ECB data level (ce1fm*n*) or DECB (idecfa) before issuing attac_id.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Pointer to the start of the storage address of the newly attached block.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example detaches a storage block from ECB level 4 (D4) using detac_id. Another block is obtained on level 4 and also detached using detac_id. Then the first block is attached using attac_id. Note that the file address is saved and then re-stored on the appropriate level before issuing the attac_id.

```
#include <tpfeq.h>
#include <tpfapi.h>

double   farw_save1;
double   farw_save2;

getfc(D4,GETFC_PRIME,"OM",GETFC_BLOCK,GETFC_NOSERRC);
memcpy(&farw_save1,&ecbptr()->ebcfw4,sizeof(double));
detac_id(D4);

getfc(D4,GETFC_PRIME,"ZJ",GETFC_BLOCK,GETFC_SERRC);
memcpy(&farw_save2,&ecbptr()->ebcfw4,sizeof(double));
detac_id(D4);

memcpy(&ecbptr()->ebcfw4,&farw_save1,sizeof(double));
attac_id(D4);
```

This example reattaches a previously detached storage block to a DECB.

```
#include <tpfeq.h>
#include <tpfapi.h>

struct cm1cm *cm;
TPF_DECB     *decb;
  ⋮
cm = (struct cm1cm *)attac_id(decb);
```

Related information

2.18, “detac_id — Detach a storage block” on page 55.
2.3, “attac — Attach a previously detached storage block” on page 30.
2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.

2.6 comic — Get communication resource information

TPF compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void *comic(enum comic_type type, enum comic_data data,
           (void *) id_ptr, (void *) info_ptr, int length);
```

type
Identifies the type of communication resource identifier that you are specifying in the *id_ptr* parameter. Use one of the following:

COMIC_CRI
id_ptr points to a 3-byte communication resource identifier (CRI).

COMIC_CRN
id_ptr points to an 8-byte communication resource name (CRN).

COMIC_SLCID
id_ptr points to a 8-byte ITA/IATA synchronous link control identifier (SLC-ID).

data
Specifies whether you want ALCS to access the system data, or the user data part of the communication table:

COMIC_SYS
Get data from the system part of the communication table. This table is defined by the co0ic structure. (See 1.12, "<c$co0ic.h>" on page 10.)

COMIC_USER
Get data from the user part of the communication table.

id_ptr
Pointer to a CRI, a CRN, or a SLC_ID. (The *type* parameter specifies which.)

info_ptr
Pointer to the area to receive the communication table information. You can specify NULL for this parameter, in which case ALCS does not move the communication resource information to a specified area.

length
The length of the area specified by *info_ptr*.

Description

Use the comic function to get details of a communication resource. You must provide comic with a system data area of type co0ic. This structure is defined in <c$co0ic.h>.

Normal return

Pointer to the requested data.

If you provide an area to receive the data (by setting up *info_ptr*), `comic` copies the communication resource data to this area. If you specify *info_ptr* as NULL, `comic` returns the information in a location that might be overwritten the next time any entry calls `comic`.

Error return

If the communication resource does not exist, `comic` returns NULL.

If the length specified by the *length* parameter is too small to hold the data supplied by the `comic` function, the function returns a value of -1. `comic` puts the specified amount of data into the *info_ptr* area, the remainder is truncated.

Any other negative value indicates an error.

Loss of control

This function does not cause the entry to lose control.

Example

This example gets the system data for the terminal whose CRN ID is PYES0740 and displays the CRI.

```
#include <tpfeq.h>
#include <tpfapi.h>

char crn[8];
struct co0ic *p;

memcpy(crn,"PYES0740",8);

p  = comic(COMIC_CRN,
     COMIC_SYS,
     crn,
     NULL,
     0);

if ( (int)p > 0  && p != NULL )
 {
    printf("CRI is %06x\n", p->icecri);
    if (p->icescst)
       printf("Resource is in session with ALCS\n");
 }
```

Related information

1.12, “<c$co0ic.h>” on page 10.

2.7 corhc — Define and hold a resource

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void corhc(void *resource);
```

resource
Pointer to the resource (an area of storage) for which you are requesting exclusive control.

Description

Use the corhc function to request exclusive control over a resource. You could use corhc, for example, to update a storage area such as a global record that is shared across processors in a multiprocessing environment.

When the entry has finished with the exclusive use of the resource, you must call the function coruc to free it for use with other entries.

Normal return

Void. ALCS gives the entry exclusive control over the resource.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control. (ALCS does not return control to the entry until it can give the entry exclusive control over the resource.)

Example

This example obtains exclusive control of the global record _Q05MET based on the address of the record:

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfglbl.h>

struct msgexp *q05met;
  ⋮
q05met = (struct msgexp *) glob(_Q05MET);
corhc(q05met);
```

Related information

2.8, “coruc — Unhold a resource” on page 38.
2.49, “glob — Address an application global field or record” on page 124.
2.50, “global — Operate on an application global field” on page 125.

2.8 coruc — Unhold a resource

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void coruc(void *resource);
```

resource
Pointer to the resource (area of storage) for which you are requesting ALCS to release your exclusive control.

Description

Use the coruc function to release exclusive control over a resource.

The entry must only release control of a resource over which it has established exclusive control by a previous corhc function call.

Normal return

Void. ALCS has removed exclusive control over the resource from the entry.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example obtains a hold on the global record _Q05MET based on the address of the record, and then releases it:

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfglbl.h>

struct msgexp *q05met;
  ⋮
q05met = (struct msgexp *) glob(_Q05MET);
corhc(q05met);
  ⋮
coruc(q05met);
```

Related information

2.7, “corhc — Define and hold a resource” on page 37.
2.49, “glob — Address an application global field or record” on page 124.
2.50, “global — Operate on an application global field” on page 125.

2.9 credc — Create an entry for deferred scheduling

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void credc(int length, void *data, void *program_name);
```

length
The number of bytes of data you want to be passed to the created entry. The minimum value is zero (indicating that no data is to be passed); the maximum value is 112.

TPF compatibility

In TPF, the maximum value is 104.

data
Pointer to the data to be passed to the new entry.

program_name
Pointer to a 4-character program name. ALCS uses this program to process the created entry.

The program name can be coded with or without quotes.

TPF compatibility

The program name should be coded without quotes for programs which must be compatible with TPF.

See 2.1, “Common parameters” on page 25.

Description

Use the credc function to create a new, independent, entry for deferred processing.

You can transfer up to 112 bytes of data to the new entry. Specify this data using the *length* and *data* parameters. ALCS copies this data to the work area ebw000 of the created entry's ECB.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example creates a deferred entry for program PGM0. It passes the string “VPH” as input data to the program:

```
#include <tpfeq.h>
#include <tpfapi.h>
#define PGM0 "PGM0"

char *parmstring = "VPH";
  ⋮
credc(strlen(parmstring),parmstring,PGM0);
```

Related information

2.10, “creec — Create an entry with an attached storage block” on page 41.
2.11, “cremc — Create an entry for immediate scheduling” on page 43.
2.12, “cretc — Create an entry for scheduling after a time delay” on page 45.
2.13, “crexc — Create an entry for deferred scheduling” on page 47.

2.10 creec — Create an entry with an attached storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void creec(int length, void *data, void *program_name, enum t_lvl level,
           int priority);
```

length
The number of bytes of data you want to be passed to the created entry. The minimum value is zero (indicating that no data is to be passed); the maximum value is 112.

TPF compatibility

In TPF, the maximum value is 104.

data
Pointer to the data to be passed to the new entry.

program_name
Pointer to a 4-character program name. ALCS uses this program to process the created entry.

The program name can be coded with or without quotes.

TPF compatibility

The program name should be coded without quotes for programs which must be compatible with TPF.

See 2.1, “Common parameters” on page 25.

level
A storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. You want the storage block on this level to be passed to the newly created entry.

priority
Use the value CREEC_DEFERRED to request ALCS to give the new entry defer list priority, and CREEC_IMMEDIATE to give it ready list priority.

Description

Use the creec function to create a new, independent, entry with an attached storage block copied from the calling entry. You can request immediate or deferred processing of the new entry.

The storage block that ALCS passes to the new entry is the one attached at the specified storage level in the ECB of the calling entry. ALCS attaches a copy of this block at storage level zero (D0) of the created entry’s ECB.

In addition to the storage block, you can transfer up to 112 bytes of data to the new entry. You specify this data using the *length* and *data* parameters. ALCS copies this data to the work area ebw000 of the created entry’s ECB.

ALCS releases the storage block that it has copied to the new entry. It is no longer available to the calling program.

Attention

This function can lead to poor response times when used to create too many new entries. See 2.54, "lodic — Get system load information" on page 134 and the *ALCS Application Programming Guide*.

Normal return
Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Example
This example creates an entry to be processed immediately by program OMA0. ALCS passes a copy of the storage block on level 5 (D5) to level 0 (D0) of the new entry's ECB. It also passes the string "755/15AUG" to the work area of the new entry's ECB.

```
#include <tpfeq.h>
#include <tpfapi.h>
#define OMA0 "OMA0"

char *parmstring = "755/15AUG";
  ⋮
creec(strlen(parmstring),parmstring,OMA0,D5,CREEC_IMMEDIATE);
```

Related information
2.9, "credc — Create an entry for deferred scheduling" on page 39.
2.11, "cremc — Create an entry for immediate scheduling" on page 43.
2.12, "cretc — Create an entry for scheduling after a time delay" on page 45.
2.13, "crexc — Create an entry for deferred scheduling" on page 47.
2.54, "lodic — Get system load information" on page 134.

2.11 cremc — Create an entry for immediate scheduling

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void cremc(int length, void *data, void *program_name);
```

length
The number of bytes of data you want to be passed to the created entry. The minimum value is zero (indicating that no data is to be passed); the maximum value is 112.

TPF compatibility

In TPF, the maximum value is 104.

data
Pointer to the data you want to be passed to the new entry.

program_name
Pointer to a 4-character program name. ALCS uses this program to process the created entry.

The program name can be coded with or without quotes.

TPF compatibility

The program name should be coded without quotes for programs which must be compatible with TPF.

See 2.1, “Common parameters” on page 25.

Description

Use the cremc function to create a new, independent, entry for immediate processing.

You can transfer up to 112 bytes of data to the new entry. You specify this data using the *length* and *data* parameters. ALCS copies this data to the work area ebw000 of the created entry's ECB.

Attention

This function can lead to poor response times when used to create too many new entries. See 2.54, “lodic — Get system load information” on page 134 and the *ALCS Application Programming Guide*.

Normal return

Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Example
This example creates an entry for immediate processing by program PGM0. It passes the string “VPH” as input to the program:

```
#include <tpfeq.h>
#include <tpfapi.h>
#define PGM0 "PGM0"

char *parmstring = "VPH";
  ⋮
cremc(strlen(parmstring),parmstring,PGM0);
```

Related information
2.9, “credc — Create an entry for deferred scheduling” on page 39.
2.10, “creec — Create an entry with an attached storage block” on page 41.
2.12, “cretc — Create an entry for scheduling after a time delay” on page 45.
2.13, “crexc — Create an entry for deferred scheduling” on page 47.
2.54, “lodic — Get system load information” on page 134.

2.12 cretc — Create an entry for scheduling after a time delay

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void cretc(int type, void *program_name, int units, void *action);
```

type
Specifies whether the units parameter is expressed in minutes or seconds. Use the value CRETC_MINUTES or CRETC_SECONDS.

program_name
Pointer to a 4-character program name. ALCS uses this program to process the created entry.

The program name can be coded with or without quotes.

TPF compatibility

The program name should be coded without quotes for programs which must be compatible with TPF.

See 2.1, “Common parameters” on page 25.

units
An integer value specifying the time in minutes or seconds that is to elapse before activating the entry.

action
Pointer to 4 bytes of data. ALCS copies these 4 bytes of this data to the new entry, in the ECB, starting at ebw000.

Description

Use the cretc function to create a new, independent, entry after a specified amount of time has elapsed. You can transfer 4 bytes of data to the work area ebw000 of the created entry's ECB.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example creates a new entry for program QZZ0 after a 5 seconds delay. It stores the 4-byte character string “INIT” in the field ebw000 of the new ECB.

```
#include <tpfeq.h>
#include <tpfapi.h>
#define QZZ0 "QZZ0"

  ⋮
cretc(CRETC_SECONDS,QZZ0,5,"INIT");
```

Related information

2.9, “credc — Create an entry for deferred scheduling” on page 39.
2.10, “creec — Create an entry with an attached storage block” on page 41.
2.11, “cremc — Create an entry for immediate scheduling” on page 43.
2.13, “crexc — Create an entry for deferred scheduling” on page 47.

2.13 crexc — Create an entry for deferred scheduling

TPF compatibility

ALCS supports crexc for TPF compatibility only. In ALCS it has the same effect as credc.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void crexc(int length, void *data, void *program_name);
```

length
The number of bytes of data you want to be passed to the created entry. The minimum value is zero (indicating that no data is to be passed); the maximum value is 112.

TPF compatibility

In TPF, the maximum value is 96.

data
Pointer to the data to be passed to the new entry.

program_name
Pointer to a 4-character program name. ALCS uses this program to process the created entry. See 2.1, “Common parameters” on page 25.

Description

Use the crexc function to create a new, independent, entry for deferred processing.

You can transfer up to 112 bytes of data to the new entry. You specify this data using the *length* and *data* parameters. ALCS copies this data to the work area ebw000 of the created entry's ECB.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example creates a deferred entry for processing by program PGM0. It passes the string “VPH” as input data to the program.

```
#include <tpfeq.h>
#include <tpfapi.h>
#define PGM0 "PGM0"

char *parmstring = "VPH";
  ⋮
crexc(strlen(parmstring),parmstring,PGM0);
```

Related information

2.9, “credc — Create an entry for deferred scheduling” on page 39.
2.10, “creec — Create an entry with an attached storage block” on page 41.
2.11, “cremc — Create an entry for immediate scheduling” on page 43.
2.12, “cretc — Create an entry for scheduling after a time delay” on page 45.

2.14 crusa — Release storage blocks from specified levels

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void crusa(int count, enum t_lvl level, ...);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void crusa(int count, TPF_DECB *decb, ...);
```

count
The number of *level* or *decb* parameters included in the argument list (minimum 1, maximum 16).

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

Description

Use the crusa function to release storage blocks from an entry.

crusa releases a storage block attached at any, or all, the ECB levels specified in the *level* parameter or the DECBs specified in the *decb* parameter. crusa does not return an error if one or more of the levels or DECBs you specify does not have an attached storage block.

Note: Applications that call this function using DECBs instead of ECB levels must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example tests ECB storage levels 0, 4, and 10 (D0, D4, and DA) and any specified DECBs for storage blocks. It releases the blocks if present.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB *decb1, *decb2;
  ⋮
crusa(3,DA,D0,D4);     /* Clear levels D0, D4, DA */
  ⋮
crusa(2,decb1,decb2);  /* Clear DECBs decb1, decb2 */
```

Related information

2.53, “levtest — Test a storage level” on page 132.
2.61, “relcc — Release a storage block” on page 148.

2.15 defrc — Defer processing

TPF compatibility

Do not call defrc if the entry is holding any records (record hold) or resources (resource hold).

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void defrc(void);
```

Description

Use the defrc function to force the entry to lose control. *ALCS Concepts and Facilities* explains how ALCS dispatches entries that use the defrc and dlayc functions.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

Example

This example suspends processing of the currently executing program.

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
defrc();
```

Related information

ALCS Concepts and Facilities.
2.19, "dlayc — Delay processing" on page 57.
2.54, "lodic — Get system load information" on page 134.
2.97, "waitc — Wait for all outstanding I/O requests to complete" on page 222.

2.16 detac — Detach a storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void detac(enum t_lvl level);
```

level
A storage level (D0, D1,...DF.). See 2.1, “Common parameters” on page 25

Description

Use the detac function to detach a storage block from an entry temporarily. You can later reattach the block by calling the attac function.

Normal return

Void. The storage level is now available for use.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example detaches a storage block from level 6 (D6).

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
detac(D6);
```

Related information

2.3, “attac — Attach a previously detached storage block” on page 30.
2.18, “detac_id — Detach a storage block” on page 55.
2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.

2.17 detac_ext — Detach a storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void detac_ext(enum t_lvl level, int ext);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void detac_ext(TPF_DECB *decb, int ext);
```

level
An ECB storage level (D0, D1,...DF.). See 2.1, “Common parameters” on page 25

decb
A pointer to a data event control block (DECB).

ext
Indicate which block is to be detached. If you code more than one term for *ext*, you must separate the terms with a plus sign (+). Use any of the following terms:

DETAC_USER_DEFAULT
The blocks detached from the specified ECB level or DECB will be reattached in a last-in-first-out (LIFO) order using attac_ext(...,ATTAC_USER_DEFAULT).

DETAC_USER_ACPDB
The detached block will be reattached using attac_ext(...,ATTAC_USER_ACPDB) if the block file address matches the file address for the specified ECB level or DECB.

DETAC_CHECK
This term is provided for compatibility with TPF. ALCS ignores it.

TPF compatibility

When you use DETAC_CHECK (or DETAC_DEFAULT) in TPF, detac_ext checks that a storage block is attached at the specified ECB level or DECB and dumps with exit if not. In ALCS detac_ext ignores this option and does not check for an attached block.

DETAC_NOCHECK
Indicates that no check will be made to determine if there is a block to be detached.

DETAC_DEFAULT
Indicates that the defaults are DETAC_USER_DEFAULT and DETAC_CHECK.

Description

Use the detac_ext function to detach a storage block from an entry temporarily. You can later reattach the block by calling the attac_ext function.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void. The specified ECB storage level or DECB is now available for use.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example detaches a storage block from ECB level 6 (D6).

```
#include <tpfeq.h>
#include <tpfapi.h>
  ⋮
detac_ext(D6, DETAC_USER_DEFAULT);
```

This example detaches a storage block from a DECB.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB     *decb;
  ⋮
detac_ext(decb, DETAC_USER_ACPDB);
```

Related information

2.3, “attac — Attach a previously detached storage block” on page 30.
2.4, “attac_ext — Attach a previously detached storage block” on page 31.
2.5, “attac_id — Attach a previously detached storage block” on page 33.
2.16, “detac — Detach a storage block” on page 52.
2.18, “detac_id — Detach a storage block” on page 55.
2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.

2.18 detac_id — Detach a storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void detac_id(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void detac_id(TPF_DECB *decb);
```

level
An ECB storage level (D0, D1,...DF.). See 2.1, “Common parameters” on page 25

decb
A pointer to a data event control block (DECB).

Description

Use the detac_id function to detach a storage block from an entry temporarily. You can later reattach the block by calling the attac_id function.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void. The storage level is now available for use.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example detaches a storage block from ECB level 6 (D6).

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
detac_id(D6);
```

This example detaches a storage block from a DECB.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB     *decb;
  ⋮
detac_id(decb);
```

Related information

2.3, “attac — Attach a previously detached storage block” on page 30.
2.5, “attac_id — Attach a previously detached storage block” on page 33.
2.16, “detac — Detach a storage block” on page 52.
2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.

2.19 dlayc — Delay processing

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void dlayc(void);
```

Description

Use the dlayc function to suspend processing of the current entry. *ALCS Concepts and Facilities* explains how ALCS dispatches entries that use the dlayc and defrc functions.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

Example

This example suspends processing of the currently executing program.

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
dlayc();
```

Related information

ALCS Concepts and Facilities

2.15, "defrc — Defer processing" on page 51.
2.54, "lodic — Get system load information" on page 134.
2.97, "waitc — Wait for all outstanding I/O requests to complete" on page 222.

2.20 ecbptr — Access the entry control block

Format

```
#include <tpfeq.h>
#include <c$eb0eb.h>

struct eb0eb *ecbptr(void);
```

Description

Use the ecbptr function to return a pointer to the entry control block (ECB). The ECB structure eb0eb, is defined in <c$eb0eb.h> (#included by <tpfeq.h>) and described in *ALCS Application Programming Guide*.

Normal return

Pointer to the current entry's ECB.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example obtains a pointer to the record on level 2 (D2).

```
#include <tpfeq.h>
#include <c$eb0eb.h>
#include <c$am0sg.h>

struct am0sg *amsg;           /* Pointer to message block */
  ⋮
amsg = ecbptr()->ce1cr2;      /*  Get base of record      */
```

Related information

ALCS Application Programming Guide.

2.21 entdc — Enter a program with no return expected to any program

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void entdc(void *program_name, struct TPF_regs *regs);
```

program_name
Pointer to the name of the application program to be entered.

regs
Pointer to a TPF_regs structure (defined in <tpfeq.h>). You can set up register values which ALCS passes to the entered program (see "Description"). When you code this parameter as NULL, ALCS does not load the general registers before passing control to the entered program.

Description

Use the entdc function to transfer control to another (or the same) application program when you do not require a return to this or any previous calling program. (entdc clears any existing program nesting levels.)

Before calling entdc, you can set up data in members regs.r1 through regs.r7 of the TPF_regs structure *regs*. ALCS loads general registers 1 (RG1) through 7 (RGF) with the values you have set up in regs.r1 through regs.r7 before it passes control to the entered program.

TPF compatibility

TPF also loads general register 0 (RAC) with regs.r0.

entdc releases all auto-storage blocks, storage you obtained using calloc and malloc, stack blocks, and static blocks. You cannot pass data held in this storage to the called program.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example initializes member r1 of the TPF_regs structure regs with a 4-character message and member r2 with the address of the storage block currently attached at level 8 (D8). It calls the entdc function to transfer control to program PGMX without return. ALCS loads general registers 1 (RG1) and 2 (RGA) with the two specified values before passing control to PGMX.

```
#include <tpfeq.h>
#include <tpfapi.h>
#define PGMX "PGMX"

struct TPF_regs *regs = (struct TPF_regs *)&(ecbptr()->ebx000);
char *msg_text;
  ⋮
regs->r1 = (int)msg_text;
regs->r2 = (int)ecbptr()->ce1cr8;
entdc(PGMX,regs);
```

Related information

2.22, “entrc — Enter a program with a return expected to the calling program” on page 61.
ALCS Application Programming Guide.

2.22 entrc — Enter a program with a return expected to the calling program

TPF compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int entrc(char *program_name, struct TPF_regs *regs);
```

program_name
Pointer to the 4-character name of the application program you want to enter. See 2.1, “Common parameters” on page 25.

regs
Pointer to a TPF_regs structure (defined in <tpfeq.h>). You can set up register values which ALCS passes to the entered program (see “Description”). When you code this parameter as NULL, ALCS does not load the general registers before passing control to the entered program.

Description

Use the entrc function to enter an application program which is expected to return control back to the entering program, or to a previous program which has called entrc.

Before calling entrc, you can set up data in members regs.r0 through regs.r7 of the TPF_regs structure *regs*. ALCS loads general registers 0 (RG0) through 7 (RGF) with the values you have set up in regs.r0 through regs.r7 before it passes control to the entered program.

If the entered program is written in BAL (Basic Assembler Language) it can return by issuing the assembler BACKC macro. If the entered program is written in the C or C++ languages it should use a return statement. Before it does so, ALCS stores the current contents of general registers 0 (RG0) through 7 (RGF) in the *regs* structure members regs.r0 through regs.r7 before returning control. The entered program can return information to the entering program. The contents of the structure TPF_regs are likely to be changed by the entered program.

Normal return

The return code from the program that was entered. This makes the the outcome of the program entered visible to the caller.

Error return

Not applicable.

Loss of control

The function can cause the entry to lose control.

Example

This example enters a program named PBC1 with an expected return. It supplies the contents of general registers `regs.r1`to `regs.r7` to the entered program.

```
#include <tpfeq.h>
#include <tpfapi.h>

struct TPF_regs regs  = [0,1,2,3,4,5,6,7];
int retCode;

retCode = entrc("PBC1", &regs);
```

Related information

2.21, “`entdc` — Enter a program with no return expected to any program” on page 59.

2.23 face — Compute the file address of a fixed file record from a record type value

Format

```
#include <tpfeq.h>

void face(struct TPF_regs &regs);
```

regs
Pointer to a TPF_regs structure containing the following:

regs.r0
The record ordinal number.

regs.r6
The numeric record type.

regs.r7
Pointer to an 8-byte field where the file address is to be stored. This can be an ECB data level. face stores the file address in the last 4 bytes.

Description

Use the face function to compute the 4-byte file address of an ALCS fixed-file record. Before calling face you must set up the record ordinal number and the record type number in the TPF_regs structure *regs*, as shown above.

The record type number and maximum record ordinal number are defined by your ALCS system programmer when the ALCS database is generated. See *ALCS Installation and Customization*.

To avoid the use of hard-coded fixed-file record types, consider using facs instead of face. (See 2.24, "facs — Compute the file address of a fixed file record from a record type symbol" on page 65).

Normal return

ALCS copies the file address of the record to the area pointed to by regs.r7.

regs.r0 contains the maximum ordinal number for the record type (nonzero).
regs.r1 to regs.r5 are unchanged.
regs.r6 and regs.r7 are undefined.

Error return

The area pointed to by regs.r7.

regs.r0 contains zero.
regs.r1 to regs.r5 are unchanged.
regs.r6 is undefined.
regs.r7 contains an error code:
1 if the record type was invalid, or
2 if the record ordinal was invalid.

Loss of control

This function does not cause the entry to lose control.

Example

This example returns the file address of ordinal number 345 in the real-time fixed-file record type of 45.

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfio.h>

#define  VD1RI_NUM  45                /* fixed-file record      */
                                      /* type number            */
 regs.r0 =  345;                      /* record ordinal         */
 regs.r6 =  VD1RI_NUM;                /* fixed-file type number */
 regs.r7 = (int) & (ecbptr()->ce1faa); /* address of level A     */
 face(&regs);
 if (!regs.r0)
 {
   if (regs.r7 == 1 )
     puts("Invalid record type");
   else
     puts("Invalid ordinal number");
   abort();
 }
 printf("File address is %08X\n", ecbptr()->ebcfaa );
 findc(DA);
 if (waitc())
 {
   puts("I/O error occurred");
   abort();
 }
```

Related information

2.24, “facs — Compute the file address of a fixed file record from a record type symbol” on page 65.
2.59, “raisa — Compute the file address of a general file record” on page 144.
2.64, “ronic — Return information about a DASD record or record type” on page 153.
2.88, “tpf_fac8c — Compute an online 8-byte file address” on page 205.
2.89, “tpf_fa4x4c — Convert a file address” on page 207.

2.24 facs — Compute the file address of a fixed file record from a record type symbol

Format

```
#include <tpfeq.h>

void facs(struct TPF_regs &regs);
```

regs
Pointer to a TPF_regs structure containing the following:

regs.r0
The record ordinal number.

regs.r6
Pointer to an 8-byte string containing the symbolic record name, left adjusted, padded with space characters.

regs.r7
Pointer to an 8-byte field where the file address is to be stored. This can be an ECB data level. facs stores the file address in the last 4 bytes.

Description

Use the facs function to compute the 4-byte file address of an ALCS fixed-file record. Before calling facs you must set up the record ordinal number and a pointer to the record type symbol in the TPF_regs structure *regs*. The record type name and maximum record ordinal number are defined by your ALCS system programmer when the ALCS data base is generated. See *ALCS Installation and Customization*.

Normal return

ALCS copies the file address of the record to the area pointed to by regs.r7.

regs.r0 contains the maximum ordinal number for the record type (nonzero).
regs.r1 to regs.r5 are unchanged.
regs.r6 and regs.r7 are undefined.

Error return

The area pointed to by regs.r7.

regs.r0 contains zero.
regs.r1 to regs.r5 are unchanged.
regs.r6 is undefined.
regs.r7 contains an error code:
1 if the record type was invalid, or
2 if the record ordinal was invalid.

Loss of control

This function does not cause the entry to lose control.

Example

This example returns the file address of ordinal number 123 in the real-time fixed-file record type #VD1IR.

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfio.h>
#define  VD1RI "#VD1IR  "      /* record type name padded with blanks */

 regs.r0 =  123;                       /* record ordinal     */
 regs.r6 =  (int) VD1RI;               /* point to type name */
 regs.r7 = (int) & (ecbptr()->ce1faa); /* address of level A */
 facs(&regs);
 if (!regs.r0)
 {
   if (regs.r7 == 1 )
     puts("invalid record type");
   else
     puts("invalid ordinal number");
   abort();
 }
 printf("file address is %08X\n", ecbptr()->ebcfaa );
 findc(DA);
 if (waitc())
 {
   puts("I/O error occurred");
   abort();
 }
```

Related information

2.23, “face — Compute the file address of a fixed file record from a record type value” on page 63.
2.59, “raisa — Compute the file address of a general file record” on page 144.
2.64, “ronic — Return information about a DASD record or record type” on page 153.
2.88, “tpf_fac8c — Compute an online 8-byte file address” on page 205.
2.89, “tpf_fa4x4c — Convert a file address” on page 207.

2.25 filec — Write a DASD record

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filec(enum t_lvl level [,GDS] [,NOTAG]);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

NOTAG
If you code this parameter, when ALCS writes the record it does not insert the program name into the record header. See 2.1, “Common parameters” on page 25.

Description

Use the `filec` function to write the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the corresponding data level. (The storage block must be the same size as the record.)

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

After calling `filec`, the block of storage containing the data to be written is no longer available to the application program and the application can immediately reuse this storage level.

`filec` does not change the hold status of the file address. To write a record to a held file address, either use `filuc` or follow the `filec` function with a `unfrc` function call.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent `waitc` function. The entry cannot check if the write was successful.

Example

This example is a subroutine that calculates the address of a fixed-file record, reads the record, sets the contents of the record to binary zeros, then writes it using `filec`.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <tpfapi.h>

/*============================================================*/
/*   clear_fixed(type, ordinal)                               */
/*                                                            */
/*   subroutine to clear the contents of a fixed file record  */
/*                                                            */
/*   the first parameter is a pointer to the record type name */
/*     ( 8 characters eg. "#L3TST  ")                         */
/*                                                            */
/*   the second parameter is the record ordinal number        */
/*                                                            */
/*============================================================*/

void clear_fixed(char *type, int ordinal)
{
int  waitrc;                        /* waitc return code */
struct TPF_regs  regs;

      puts("APITEST2: filec APR-C sample");

      /*-------------------------------------------------*/
      /* set up for file address calculation             */
      /*-------------------------------------------------*/

      printf("type is %s  ordinal is %d\n", type, ordinal);
      regs.r0 = ordinal;
      regs.r6 = (int) type;
      regs.r7 = (int) & (ecbptr()->ce1faa);

      facs(&regs);         /* calculate the file address  */

      if (!regs.r0)
      {
        if (regs.r7 == 1 )
          puts("error: invalid record id");
        else
          puts("error: invalid ordinal number");

        exit(0);
      }

      printf("file address is %08X\n", ecbptr()->ebcfaa );
```

```
    /*-------------------------------------------------*/
    /* set up to read the record                       */
    /*-------------------------------------------------*/

    /* clear the record id and record code check       */
    memset(&ecbptr()->ce1faa,0,sizeof(ecbptr()->ce1faa));

    findc(DA);                   /* read the record    */

    if (waitrc = waitc())
    {
      printf("error: return code %02X from waitc\n",waitrc);
      exit(0);
    }

    printf("record size is %d bytes\n", levtest(DA) );


    /*-------------------------------------------------*/
    /* clear the record and file it                    */
    /*-------------------------------------------------*/

    /* clear the record to binary zeros                */
    memset(ecbptr()->ce1cra ,0 ,levtest(DA));

    /* clear the record id and record code check       */
    memset(&ecbptr()->ce1faa,0,sizeof(ecbptr()->ce1faa));

    filec(DA);                    /* file the record   */

    puts("record cleared to zeros and filed");

    return;
  }
⋮
clear_fixed("#L3TST  ",0);   /* clear #L3TST record ordinal 0 */
clear_fixed("#L3TST  ",1);   /* clear #L3TST record ordinal 1 */
⋮
```

Related information

2.26, “filec_ext — Write a DASD record, with extended options” on page 70.
2.27, “file_record — Write a DASD record” on page 72.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.
2.31, “filuc — Write a DASD record and unhold the file address” on page 82.

2.26 filec_ext — Write a DASD record, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filec_ext(enum t_lvl level, unsigned int ext_file);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_file
If you code the value FILE_NOTAG, when ALCS writes the record it does not insert the program name into the record header. ALCS ignores any other values (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

The filec_ext function writes the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the associated data level. (The storage block must be the same size as the record.)

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to '\0'.

After calling filec_ext, the storage block containing the data to be written is no longer available to the application program. The application can immediately reuse this storage level.

filec_ext does not change the hold status of the file address. To write a record to a held file address, either use filuc_ext or follow the filec_ext function with a unfrc function call.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the write was successful.

Example

This example writes the data in the storage block on level 7 (D7) to a general data set, bypasses the record header update, and releases the storage block.

```
#include <tpfeq.h>
#include <tpfio.h>
  ⋮
filec_ext(D7,FILE_NOTAG+FILE_GDS);
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.27, “file_record — Write a DASD record” on page 72.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.
2.31, “filuc — Write a DASD record and unhold the file address” on page 82.

2.27 file_record — Write a DASD record

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void file_record(enum t_lvl level, unsigned int *address, char *record_id,
                 unsigned char rcc, enum t_act type [,GDS] [,NOTAG]);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

address
Pointer to the file address that you want the record to be written to, or NULL.

record_id
Pointer to the 2-character record ID of the record to be written to DASD, or RECID_RESET.

rcc
The record code check (RCC) of the record to be written to DASD, or '\0'.

type
An indicator that tells ALCS whether after the write, it must:

- Release the storage block
- Unhold the record.

Use one of the following values:

NOHOLD
ALCS releases the storage block and does not change the hold status of the file address.

UNHOLD
ALCS releases the storage block and unholds the file address.

NOREL
ALCS does not release the storage block and does not change the hold status of the file address.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

NOTAG
If you code this parameter, when ALCS writes the record it does not insert the program name into the record header. See 2.1, “Common parameters” on page 25.

Description

Use the file_record function to write the contents of the storage block, attached at the level specified in the *level* parameter, to the file address specified in the *address* parameter. When the *address* parameter is NULL, file_record writes the storage block to the file address stored in the data level specified in the *level* parameter. (The storage block must be the same size as the record.)

After the call, the data level contains the file address, record ID, and record code check (RCC) for the record that has been written. However, if you code the *address* or *record_id* as NULL, ALCS does not change the file address or record ID, or both (whichever is NULL).

When you specify *type* as UNHOLD, the entry must be holding the file address. ALCS unholds it. When you specify NOREL or NOHOLD, ALCS does not change the hold status of the file address.

The record ID in the *record_id* parameter must match the record ID in the record image. You can suppress this this check by coding RECID_RESET in the *record_id* parameter.

The record code check (RCC) in the *rcc* parameter must match the RCC in the record image. You can suppress this check by setting *rcc* to `'\0'`.

When the *type* parameter contains the value NOREL, you can test for these errors by examining the return value of the subsequent `waitc` function (which is required, see “Loss of control” below). When the *type* parameter does not contain NOREL, errors cause a system error (dump) with exit.

Note: Unless you use a value of NOREL in the *type* parameter, ALCS releases the storage block.

Normal return
Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

When you use the NOHOLD and UNHOLD values for the *type* parameter, ALCS does not return the status of the operation to the application program. You cannot test the success of the write.

When you code the value NOREL in the *type* parameter, you must follow this function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return from the `waitc`, or the function containing the implied `waitc`, lets you check whether the preceding I/O operations, including this write, were successful. See *ALCS Application Programming Guide*.

Example

This example files a record from level 6 (D6) with record ID set to 'IM', the RCC set to '\0', and the file address taken from the forward chain field of another record.

```
#include <tpfeq.h>
#include <tpfio.h>

struct im0im
   {
      char               im0bid[2];          /* record ID         */
      unsigned char      im0rcc;             /* record code check */
      unsigned char      im0ctl;             /* control byte      */
      char               im0pgm[4];          /* program stamp     */
      unsigned long int  im0fch;             /* forward chain     */
      unsigned long int  im0bch;             /* backward chain    */
      short int          im0cct;             /* byte count        */

      (other data fields)
  ⋮
   } *inm;
  ⋮

file_record(D6,(unsigned int *)&(inm->im0fch),"IM",'\0',NOHOLD);
```

Related information

2.28, “file_record_ext — Write a DASD record, with extended options” on page 75.
2.35, “find_record — Read a DASD record” on page 90.
2.59, “raisa — Compute the file address of a general file record” on page 144.
2.95, “unfrc — Unhold a file address” on page 218.

2.28 file_record_ext — Write a DASD record, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void file_record_ext(enum t_lvl level, unsigned int *address, char *record_id,
                 unsigned char rcc, enum t_act type, unsigned int ext_file);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void file_record_ext(TPF_DECB *decb, TPF_FA8 *fa8, char *record_id,
                 unsigned char rcc, enum t_act type, unsigned int ext_file);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

address
Pointer to the 4-byte file address that you want the record to be written to, or NULL.

fa8
Pointer to the 8-byte file address in 4x4 format that you want the record to be written to, or NULL.

record_id
Pointer to the 2-character record ID of the record to be written to DASD, or RECID_RESET.

rcc
The record code check (RCC) of the record to be written to DASD, or `'\0'`.

type
An indicator that tells ALCS whether after the write, it must:

- Release the storage block
- Unhold the record.

Use one of the following values:

NOHOLD
ALCS releases the storage block and does not change the hold status of the file address.

UNHOLD
ALCS releases the storage block and unholds the file address.

NOREL
ALCS does not release the storage block and does not change the hold status of the file address.

ext_file
If you code the value FILE_NOTAG, when ALCS writes the record it does not insert the program name into the record header. ALCS ignores any other

values (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

When the *level* parameter is specified, The `file_record_ext` function writes the contents of the storage block, attached at that ECB data level, to the file address specified in the *address* parameter. When the *address* parameter is NULL, `file_record_ext` writes the storage block to the file address stored in the ECB data level specified in the *level* parameter. (The storage block must be the same size as the record.)

When the *decb* parameter is specified, the `file_record_ext` function writes the contents of the storage block, attached at the data level of that DECB, to the file address specified in the *fa8* parameter. When the *fa8* parameter is NULL, `file_record_ext` writes the storage block to the file address stored in the data level of the DECB specified in the *decb* parameter. (The storage block must be same size as the record.)

After the call, the data level contains the file address, record ID, and record code check (RCC) for the record that has been written. However, if you code the file address (*address* or *fa8*) or the *record_id* as NULL, ALCS does not change the file address or record ID (whichever is NULL).

When you specify *type* as UNHOLD, the entry must be holding the file address. ALCS unholds it. When you specify NOREL or NOHOLD, ALCS does not change the hold status of the file address.

The record ID in the *record_id* parameter must match the record ID in the record image. You can suppress this check by coding RECID_RESET in the *record_id* parameter.

The record code check (RCC) in the *rcc* parameter must match the RCC in the record image. You can suppress this check by setting *rcc* to `'\0'`.

When the *type* parameter contains the value NOREL, you can test for these errors by examining the return value of the subsequent `waitc` function (which is required, see “Loss of control” below). When the *type* parameter does not contain NOREL, errors cause a system error (dump) with exit.

Notes:

1. Unless you use a value of NOREL in the *type* parameter, ALCS releases the storage block.
2. Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

When you use the NOHOLD and UNHOLD values for the *type* parameter, ALCS does not return the status of the operation to the application program. You cannot test the success of the write.

When you code the value NOREL in the *type* parameter, you must follow this function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return from the `waitc`, or the function containing the implied `waitc`, lets you check whether the preceding I/O operations, including this write, were successful.

Examples
This example writes the data in the storage block attached at ECB level 7 (D7) to a general data set, bypasses the record header update, and releases the block. The record ID is `'CD'`, the RCC is `'\0'`, and the file address is taken from the forward chain field of another record.

```
#include <tpfeq.h>
#include <tpfio.h>
  ⋮
file_record_ext(D7,(unsigned int *)&(inm->im0fch),"CD",'\0',
                UNHOLD,FILE_NOTAG+FILE_GDS);
```

This example writes the data in the storage block attached at the data level of a DECB and unholds the record. The file address and record ID are specified in the DECB.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_DECB *decb;
  ⋮
file_record_ext(decb,NULL,NULL,'\0',UNHOLD,FILE_NOTAG)
```

Related information
2.27, “`file_record` — Write a DASD record” on page 72.
2.35, “`find_record` — Read a DASD record” on page 90.
2.59, “`raisa` — Compute the file address of a general file record” on page 144.
2.88, “`tpf_fac8c` — Compute an online 8-byte file address” on page 205.
2.89, “`tpf_fa4x4c` — Convert a file address” on page 207.
2.95, “`unfrc` — Unhold a file address” on page 218.
2.96, “`unfrc_ext` — Unhold a file address, with extended options” on page 220.

2.29 filnc — Write a DASD record and retain the attached storage block

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filnc(enum t_lvl level [,GDS] [,NOTAG]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

NOTAG
If you code this parameter, when ALCS writes the record it does not insert the program name into the record header. See 2.1, “Common parameters” on page 25.

Description

Use the `filnc` function to write the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the associated data level. (The storage block must be the same size as the record.)

`filnc` does not release the storage block after the write.

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

You can test for either of these errors by examining the return of the `waitc` function. See “Loss of control”.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return value of the `waitc`, or the function containing an implied `waitc`, lets you check whether the preceding I/O operations, including this write, were successful. See *ALCS Application Programming Guide*.

Example

This example writes the data in the storage block on level 15 (DF) to the file address specified on data level 15 (DF). The storage block remains attached to storage level 15 (DF).

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
filnc(DF);
if (waitc())
  {
   errno = 0x1234;
   perror("Write error on level DF");
   abort();
  }
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.30, “filnc_ext — Write a DASD record and retain the attached storage block, with extended options” on page 80.
2.27, “file_record — Write a DASD record” on page 72.
2.31, “filuc — Write a DASD record and unhold the file address” on page 82.

2.30 filnc_ext — Write a DASD record and retain the attached storage block, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filnc_ext(enum t_lvl level, unsigned int ext_file);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_file
If you code the value FILE_NOTAG, when ALCS writes the record it does not insert the program name into the record header. ALCS ignores any other values (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

The `filnc_ext` function writes the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the associated data level. (The storage block must be the same size as the record.)

`filnc_ext` does not release the storage block after the write.

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

You can test for either of these errors by examining the return of the `waitc` function. See “Loss of control”

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return value of the `waitc`, or the function containing an implied `waitc`, lets you check whether the preceding I/O operations, including this write, were successful. See *ALCS Application Programming Guide*.

Example

This example writes the data in the storage block attached at level 7 (D7) to a general data set and bypasses the record header update. The storage block remains attached to storage level 7 (D7).

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
filnc_ext(D7,FILE_NOTAG+FILE_GDS);
if (waitc())
  {
   errno = 0x1234;
   perror("Write error on level D7");
   abort();
  }
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.27, “file_record — Write a DASD record” on page 72.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.
2.31, “filuc — Write a DASD record and unhold the file address” on page 82.

2.31 filuc — Write a DASD record and unhold the file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filuc(enum t_lvl level [,GDS] [,NOTAG]);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

NOTAG
If you code this parameter, when ALCS writes the record it does not insert the program name into the record header. See 2.1, “Common parameters” on page 25.

Description

Use the `filuc` function to write the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the associated data level. (The storage block must be the same size as the record.)

`filuc` unholds the file address after the write.

The entry must be holding the file address when you call `filuc`.

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

After calling `filuc`, the storage block containing the data to be written is no longer available to the application program. The application can immediately reuse this storage level.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent `waitc` function. The entry cannot check if the write was successful.

Example

This example writes the data in the storage block on level 4 (D4) to file, and releases the storage block. On return, the file copy of the record is available to other ECBs.

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
filuc(D4);
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.27, “file_record — Write a DASD record” on page 72.
2.32, “filuc_ext — Write a DASD record and unhold the file address, with extended options” on page 84.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.

2.32 filuc_ext — Write a DASD record and unhold the file address, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void filuc_ext(enum t_lvl level, unsigned int ext_file);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_file
If you code the value FILE_NOTAG, when ALCS writes the record it does not insert the program name into the record header. ALCS ignores any other values (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

The filuc_ext function writes the contents of the storage block, attached at the level specified in the *level* parameter, to the file address stored in the associated data level. filuc_ext unholds the file address after the write. (The storage block must be the same size as the record.)

The entry must be holding the file address when you call filuc_ext.

The record ID on the level that you specify must match the record ID in the record image.

The record code check (RCC) on the level that you specify must match the RCC in the record image. You can suppress this check by setting the RCC on the level you specify to '\0'.

After calling filuc_ext, the block of storage containing the data to be written is no longer available to the application program. The application can immediately reuse this storage level.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the write was successful.

Example

This example writes the data in the storage block attached at level 7 (D7) to a general data set, bypasses the record header update, and releases the block.

```
#include <tpfeq.h>
#include <tpfio.h>
  ⋮
filuc_ext(D7,FILE_GDS+FILE_NOTAG);
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.31, “filuc — Write a DASD record and unhold the file address” on page 82.
2.27, “file_record — Write a DASD record” on page 72.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.

2.33 findc — Read a DASD record

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void findc(enum t_lvl level [,GDS]);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the `findc` function to read the record whose file address is in the data level specified by the *level* parameter. `findc` gets a suitably sized storage block, reads the record into it, and attaches this block to the storage level you specify in the *level* parameter. (There must be no storage block already attached at this level.)

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

You can test for both these errors by using the `waitc` function (see “Loss of control”).

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return value of the `waitc` lets you check whether the preceding I/O operations, including this read, were successful. See *ALCS Application Programming Guide*.

Example

This example calculates the address of a fixed-file record on level 2 (D2) using `facs`. It then reads the record and tests for the I/O to complete If an error occurs it takes a dump, number 001234, and terminates abnormally.

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfio.h>
#define VD1RI "#VD1RI  "

/* set up storage */
struct TPF_regs *regs = (struct TPF_regs *) &(ecbptr()->ebx000);
  ⋮
regs->r6 = (int) VD1RI;               /* record ID                     */
regs->r7 = (int) &(ecbptr()->ce1fa2); /* level where file addr. goes   */
regs->r0 = 10;                        /* ordinal number                */
facs(regs);                           /* calculate fixed-file address  */
if (!regs->r0)                        /* FACS error?                   */
  if (regs->r7 == 1)                  /* invalid record ID             */
    exit(0x12345);
  else
       {                              /* invalid ordinal number        */
        errno = 0x1234;
        perror("Invalid ordinal number");
        abort();
       }

findc(D2);                            /* find the record               */
if (waitc())                          /* abort on any errors           */
  {
   errno = 0x1234;
   perror("Error reading record on level D2");
   abort();
  }
```

Related information

2.34, “`findc_ext` — Read a DASD record, with extended options” on page 88.
2.35, “`find_record` — Read a DASD record” on page 90.
2.37, “`finhc` — Read a DASD record and hold the file address” on page 97.
2.39, “`finwc` — Read a DASD record and wait for I/O completion” on page 101.
2.41, “`fiwhc` — Read a DASD record, hold the file address, and wait for I/O completion” on page 105.

2.34 findc_ext — Read a DASD record, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void findc_ext(enum t_lvl level, unsigned int ext_find);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_find
ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25. See 2.1, “Common parameters” on page 25.

Description

Use the findc_ext function to read the record whose file address is in the data level specified by the *level* parameter. findc_ext gets a suitably sized storage block, reads the record into it, and attaches this block to the storage level you specify in the *level* parameter. (There must be no storage block already attached at this level.)

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to '\0'.

You can test for both these errors by using the waitc function (see “Loss of control”).

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a waitc call, or with a call to a function that has an implied waitc. The return value of the waitc lets you check whether the preceding I/O operations, including this read, were successful. See *ALCS Application Programming Guide*.

Example

This example reads a record from a general data set onto level 7 (D7). The program has already computed the file address using `raisa` and it is on data level 7 (D7).

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
findc_ext(D7,FIND_GDS);
```

Related information

2.33, "`findc` — Read a DASD record" on page 86.
2.35, "`find_record` — Read a DASD record" on page 90.
2.37, "`finhc` — Read a DASD record and hold the file address" on page 97.
2.39, "`finwc` — Read a DASD record and wait for I/O completion" on page 101.
2.41, "`fiwhc` — Read a DASD record, hold the file address, and wait for I/O completion" on page 105.

2.35 find_record — Read a DASD record

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *find_record(enum t_lvl level, unsigned int *address,
                  char *record_id, unsigned char rcc, enum t_act type [,GDS]);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

address
Pointer to the file address from where you want the record to be read.

record_id
Pointer to the 2-character record ID of the record to be read, or RECID_RESET.

rcc
The record code check (RCC) of the record to be read, or '\0'.

type
An indicator that tells ALCS whether or not you want ALCS to hold the file address of the record. Use one of the defined values:

NOHOLD
Do not hold the file address of the record.

HOLD
Hold the file address of the record. The entry must unhold the file address before it exits.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the find_record function to read the record from the file address specified in the *address* parameter into a storage block. find_record attaches this storage block on the storage level you specify in the *level* parameter. (There must be no storage block already attached at this level.)

find_record updates the data level specified in the *level* parameter with file address, record ID, and record code check (RCC) supplied in the function parameters. It then initiates a read of the record and suspends the entry until the read is complete.

The record ID that you specify in the *record_id* parameter must match the record ID in the record at the specified file address. You can suppress this check by setting *record_id* to RECID_RESET.

The record code check (RCC) that you specify in the *rcc* parameter must match the RCC in the record at the the specified file address. You can suppress this check by setting *rcc* to '\0'.

You test for both these errors by checking that the return value of the function is NULL (see “Loss of control” below).

Ensure that the storage level that you specify in the *level* parameter is free before calling `find_record`.

If the entry has already initialized the specified data level (for example, by a previous call of the `face` or `facs` functions) you can bypass ALCS's initialization of the data level. To do so, code the *record_id* and *address* parameters as NULL.

You can use `find_record` to read a record from a general file. The entry must have calculated the file address using the `raisa` function. When reading from a general file, code *address* as NULL. (The `raisa` function puts the file address stored in the data level.)

Normal return

Pointer to the storage block containing the DASD record.

Error return

NULL if this or any outstanding I/O operation was unsuccessful.

Detailed error information is in the detail error byte for data level *n* (`ecbptr()->ce1sud[`*n*`]`). See *ALCS Application Programming Guide*.

Loss of control

The function causes the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return” above.

Example

This example reads a forward chain IM record on level 6 (D6), with hold. It checks that the record ID is `'IM'` but does not check the RCC.

```
#include <tpfeq.h>
#include <tpfio.h>

struct im0im
   {
      char              im0bid[2];          /* record ID           */
      unsigned char     im0rcc;             /* record code check   */
      unsigned char     im0ctl;             /* control byte        */
      char              im0pgm[4];          /* program stamp       */
      unsigned long int im0fch;             /* forward chain       */
      unsigned long int im0bch;             /* backward chain      */
      short int         im0cct;             /* byte count          */

      (data fields)

   } *inm, *inm_chain;
  ⋮
inm = ecbptr()->ce1cr5;             /* Base first in chain          */
if(!(inm_chain = find_record(D6,(unsigned int *)&(inm->im0fch),"IM",'\0',HOLD)))
   exit(0x12345);                   /* Issue dump & exit if error   */
else
                                    /* Normal processing path       */
  ⋮
```

Related information

2.36, “find_record_ext — Read a DASD record, with extended options” on page 93.
2.25, “filec — Write a DASD record” on page 67.
2.27, “file_record — Write a DASD record” on page 72.
2.59, “raisa — Compute the file address of a general file record” on page 144.
2.95, “unfrc — Unhold a file address” on page 218.

2.36 find_record_ext — Read a DASD record, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *find_record_ext(enum t_lvl level, unsigned int *address,
                  char *record_id, unsigned char rcc,
                  enum t_act type, unsigned int ext_find);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void *find_record_ext(TPF_DECB *decb, TPF_FA8 *fa8,
                  char *record_id, unsigned char rcc,
                  enum t_find_decb find_type, unsigned int ext_find);
```

level
An ECB level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

address
Pointer to the 4-byte file address from where you want the record to be read.

fa8
Pointer to the 8-byte file address in 4x4 format from where you want the record to be read.

record_id
Pointer to the 2-character record ID of the record to be read, or RECID_RESET.

rcc
The record code check (RCC) of the record to be read, or '\0'.

type
An indicator that tells ALCS whether or not you want ALCS to hold the file address of the record. Use one of the defined values:

NOHOLD
Do not hold the file address of the record.

HOLD
Hold the file address of the record. The entry must unhold the file address before it exits.

find_type
An indicator that tells ALCS whether or not you want ALCS to hold the file address of the record, and whether or not you want ALCS to wait for the I/O operation to complete. Use one of the defined values:

NOHOLD_NOWAIT
Do not hold the file address of the record and do not wait for the I/O operation to complete before returning to the calling program.

HOLD_NOWAIT
: Hold the file address of the record and do not wait for the I/O operation to complete before returning to the calling program. The entry must unhold the file address before it exits.

NOHOLD_WAIT
: Do not hold the file address of the record and wait for the I/O operation to complete before returning to the calling program.

HOLD_WAIT
: Hold the file address of the record and wait for the I/O operation to complete before returning to the calling program. The entry must unhold the file address before it exits.

ext_find
: ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

When the *level* parameter is specified, the `find_record_ext` function reads the record from the file address specified in the *address* parameter into a storage block. `find_record_ext` attaches this storage block on the ECB storage level specified in the *level* parameter.

`find_record_ext` updates the ECB data level specified in the *level* parameter with the file address, record ID, and record code check (RCC) supplied in the function parameters. It then initiates a read of the record and suspends the entry until the read is complete.

When the *decb* parameter is specified, the `find_record_ext` function reads the record from the file address specified in the *fa8* parameter into a storage block. `find_record_ext` attaches this storage block on the storage level of the DECB specified in the *decb* parameter.

`find_record_ext` updates the data level of the DECB specified in the *decb* parameter with the file address, record ID, and record code check (RCC) supplied in the function parameters and then initiates a read of the record. When the *find_type* parameter is NOHOLD_WAIT or HOLD_WAIT `find_record_ext` suspends the entry until the read is complete. When the *find_type* parameter is NOHOLD_NOWAIT or HOLD_NOWAIT, `find_record_ext` does not wait until the read is complete before returning to the calling program. You must not make any reference to the storage level in the DECB until you have requested the `waitc` function (see "Loss of control" below).

The record ID that you specify in the *record_id* parameter must match the record ID in the record at the specified file address. You can suppress this check by setting *record_id* to RECID_RESET.

The record code check (RCC) that you specify in the *rcc* parameter must match the RCC in the record at the specified file address. You can suppress this check by setting *rcc* to `'\0'`.

When you specify the *type* parameter or you specify the *find_type* parameter as NOHOLD_WAIT or HOLD_WAIT, you can test for both these errors by checking that the return value of the function is NULL (see “Error return” below).

When you specify the *find_type* parameter as NOHOLD_NOWAIT or HOLD_NOWAIT, you can test for these errors by examining the return value of the subsequent `waitc` function (which is required, see "Loss of control" below).

If the entry has already initialized the specified data level (for example, by a previous call of the `face`, `facs`, `tpf_fac8c`, `raisa`, `gdsnc` or `gdsrc` functions) you can bypass ALCS's initialization of the data level. To do so, code the *record_id* and *address* or *fa8* parameters as NULL.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

When you specify the *type* parameter or you specify the *find_type* parameter as NOHOLD_WAIT or HOLD_WAIT, normal return is a pointer to the storage block containing the DASD record.

When you specify the *find_type* parameter as NOHOLD_NOWAIT or HOLD_NOWAIT, normal return is NULL.

Error return

When you specify the *type* parameter or you specify the *find_type* parameter as NOHOLD_WAIT or HOLD_WAIT, error return is NULL if this or any outstanding I/O operation was unsuccessful.

Detailed error information is in the detail error byte for ECB data level *n* (`ecbptr()->ce1sud[`*n*`]`), or the DECB (`idecsud`). See *ALCS Application Programming Guide*.

When you specify the *find_type* parameter as NOHOLD_NOWAIT or HOLD_NOWAIT, error return is not applicable.

Loss of control

The function can cause the entry to lose control.

When you specify the *type* parameter or you specify the *find_type* parameter as NOHOLD_WAIT or HOLD_WAIT, the return value of the function tells the application whether the read was successful.

When you specify the *find_type* parameter as NOHOLD_NOWAIT or HOLD_NOWAIT, you must follow the function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return from the `waitc`, or the function containing the implied `waitc`, lets you check whether the preceding I/O operations, including this find, were successful.

Examples

This example reads a record from a general data set. It checks that the record ID is `'CD'` but does not check the RCC. The storage block will be attached to ECB level D7.

```
#include <tpfeq.h>
#include <tpfio.h>

unsigned file_ptr;
  ⋮
ecbptr()->ecbfa7 = 0x0d0002f5;      /* general file address to read */

find_record_ext(D7,&ecbptr()->ecbfa7,"CD",'\0',NOHOLD,FIND_GDS);
```

This example uses the tpf_fac8c function to obtain the file address of the fixed-file record with type #XMPRI and record ordinal 8. It then reads and holds the record. It does not check the record id or RCC. The storage block will be attached to the specified DECB.

Note: The record type name is padded with blanks.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_DECB *decb;
TPF_FAC8 fac8parms;
unsigned int fileptr;
  ⋮
fac8parms.ifacord = 8;
memcpy(fac8parms.ifacrec, "#XMPRI ", sizeof(fac8parms.ifacrec));
fac8parms.ifactyp = IFAC8FCS;

if ( tpf_fac8c(&fac8parms) == TPF_FAC8_NRM )
{
  /* normal return */
  fileptr = (unsigned int)find_record_ext(decb,
            &fac8parms.ifacadr, RECID_RESET, '\0', HOLD_WAIT);
}
else
{
  /* error return */
}
```

Related information

2.25, “filec — Write a DASD record” on page 67.
2.36, “find_record_ext — Read a DASD record, with extended options” on page 93.
2.44, “gdsnc — Open or close a general data set” on page 110.
2.45, “gdsrc — Compute the file address of a general data set record” on page 113.
2.59, “raisa — Compute the file address of a general file record” on page 144.
2.88, “tpf_fac8c — Compute an online 8-byte file address” on page 205.
2.89, “tpf_fa4x4c — Convert a file address” on page 207.
2.95, “unfrc — Unhold a file address” on page 218.
2.96, “unfrc_ext — Unhold a file address, with extended options” on page 220.
2.97, “waitc — Wait for all outstanding I/O requests to complete” on page 222.

2.37 finhc — Read a DASD record and hold the file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void finhc(enum t_lvl level [,GDS]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter. See 2.1, “Common parameters” on page 25.

Description

Use the finhc function to read the record whose file address is in the data level specified by the *level* parameter. finhc gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

finhc also holds the file address.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to '\0'.

You can test for both these errors by using the waitc function (see “Loss of control” below).

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a waitc call, or with a call to a function that has an implied waitc. The return value of the waitc lets you check whether the preceding I/O operations, including this read, were successful. See *ALCS Application Programming Guide*.

Example

This example reads a record from the file address specified on data level 2 (D2). The file address is held when the record is read.

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
finhc(D2);
if (waitc())
  {
    errno = 0x1234;
    perror("Error reading record on level D2");
    abort();
  }
```

Related information

2.33, “findc — Read a DASD record” on page 86.
2.35, “find_record — Read a DASD record” on page 90.
2.38, “finhc_ext — Read a DASD record and hold the file address, with extended options” on page 99.
2.39, “finwc — Read a DASD record and wait for I/O completion” on page 101.
2.41, “fiwhc — Read a DASD record, hold the file address, and wait for I/O completion” on page 105.

2.38 finhc_ext — Read a DASD record and hold the file address, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void finhc_ext(enum t_lvl level, unsigned int ext_find);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_find
ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

The finhc_ext function reads the record whose file address is in the data level specified by the *level* parameter. finhc gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

finhc_ext also holds the file address.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to '\0'.

You can test for both these errors by using the waitc function (see “Loss of control” below).

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a waitc call, or with a call to a function that has an implied waitc. The return value of the waitc lets you check whether the preceding I/O operations, including this read, were successful. See *ALCS Application Programming Guide*.

Example

This example reads a record from a general data set onto level 7 (D7). The file address has already been computed and is on data level 7 (D7). The file address is held when the record is read.

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
finhc_ext(D7,FIND_GDS);
if (waitc())
  {
    errno = 0x1234;
    perror("Error reading record on level D7");
    abort();
  }
```

Related information

2.33, “findc — Read a DASD record” on page 86.
2.35, “find_record — Read a DASD record” on page 90.
2.37, “finhc — Read a DASD record and hold the file address” on page 97.
2.39, “finwc — Read a DASD record and wait for I/O completion” on page 101.
2.41, “fiwhc — Read a DASD record, hold the file address, and wait for I/O completion” on page 105.

2.39 finwc — Read a DASD record and wait for I/O completion

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *finwc(enum t_lvl level [,GDS]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the finwc function to initiate a read of the record whose file address is stored in the specified data level. finwc gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

ALCS suspends the entry until the read is complete.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to '\0'.

You test for both these errors by checking that the return value of the function is NULL (see “Loss of control”).

Normal return

Pointer to the storage block containing the DASD record.

Error return

NULL if this or any outstanding I/O operation was unsuccessful.

Detailed error information is provided in the error byte for data level *n* (ecbptr()->ce1sud[*n*]). See *ALCS Application Programming Guide*.

Loss of control

This function can cause the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return” above.

Example

This example reads a record from file on level 2 (D2). The program has already computed the file address and stored it in data level 2 (D2).

ALCS returns control to the application program when the I/O is complete, and the record has been attached to the specified level.

```
#include <tpfeq.h>
#include <tpfio.h>

struct im0im *inm;
  ⋮
inm = finwc(D2);
```

Related information

2.33, "findc — Read a DASD record" on page 86.
2.35, "find_record — Read a DASD record" on page 90.
2.37, "finhc — Read a DASD record and hold the file address" on page 97.
2.40, "finwc_ext — Read a DASD record and wait for I/O completion, with extended options" on page 103.
2.41, "fiwhc — Read a DASD record, hold the file address, and wait for I/O completion" on page 105.

2.40 finwc_ext — Read a DASD record and wait for I/O completion, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *finwc_ext(enum t_lvl level, unsigned int ext_find);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_find
ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

The `finwc_ext` function initiates a read of the record whose file address is in the specified data level. `finwc_ext` gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

ALCS suspends the entry until the read is complete.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

You test for both these errors by checking that the return value of the function is NULL.

Normal return

Pointer to the storage block containing the DASD record.

Error return

NULL if this or any outstanding I/O operation was unsuccessful.

Detailed error information is provided in the error byte for data level *n* (`ecbptr()->ce1sud[n]`). See *ALCS Application Programming Guide*.

Loss of control

This function can cause the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return” above.

Example

This example reads a record from a general data set onto level 7 (D7). The program has already computed the file address, and stored it on data level 7 (D7).

ALCS returns control to the application program when the I/O is complete, and the record has been attached to the specified level.

```
#include <tpfeq.h>
#include <tpfio.h>

inm = finwc_ext(D7,FIND_GDS);
```

Related information

2.33, "findc — Read a DASD record" on page 86.
2.35, "find_record — Read a DASD record" on page 90.
2.37, "finhc — Read a DASD record and hold the file address" on page 97.
2.39, "finwc — Read a DASD record and wait for I/O completion" on page 101.
2.41, "fiwhc — Read a DASD record, hold the file address, and wait for I/O completion" on page 105.

2.41 fiwhc — Read a DASD record, hold the file address, and wait for I/O completion

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *fiwhc(enum t_lvl level [,GDS]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

GDS
ALCS ignores this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the fiwhc function to read the record whose file address is in the data level that you specify in the *level* parameter. fiwhc gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

ALCS suspends the entry until the read is complete. fiwhc holds the file address of the record that it reads.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to '\0'.

You test for both these errors by checking that the return value of the function is NULL.

Normal return

Pointer to the storage block containing the retrieved record image.

Error return

NULL if this, or any outstanding I/O operation was unsuccessful.

Detailed error information is contained in the error byte for data level *n* (ecbptr()->ce1sud[*n*]). See *ALCS Application Programming Guide*.

Loss of control

This function can cause the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return” above.

Example

This example reads a record from file on level 2 (D2) with hold. The program has already computed the file address and it is on data level 2 (D2).

ALCS returns control to the application program when the I/O is complete, and the record has been attached to the specified level.

```
#include <tpfeq.h>
#include <tpfio.h>

inm = fiwhc(D2);

if (inm==NULL)
{
  errno=0x1234;
  perror("Error reading record on level D2");
  abort();
}
```

Related information

2.33, “findc — Read a DASD record” on page 86.
2.35, “find_record — Read a DASD record” on page 90.
2.37, “finhc — Read a DASD record and hold the file address” on page 97.
2.39, “finwc — Read a DASD record and wait for I/O completion” on page 101.
2.42, “fiwhc_ext — Read a DASD record, hold the file address, and wait for I/O completion, with extended options” on page 107.

2.42 fiwhc_ext — Read a DASD record, hold the file address, and wait for I/O completion, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void *fiwhc_ext(enum t_lvl level, unsigned int ext_find);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

ext_find
ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the `fiwhc_ext` function to read the record whose file address is in the data level that you specify in the *level* parameter. `fiwhc_ext` gets a suitably sized storage block, reads the record into it, and attaches this block to the associated storage level. (There must be no storage block already attached at the specified storage level.)

ALCS suspends the entry until the read is complete. `fiwhc_ext` holds the file address of the record that it reads.

The record ID on the level that you specify must match the record ID in the record at the specified file address. You can suppress this check by setting the record ID to binary zero.

The record code check (RCC) on the level that you specify must match the RCC in the record at the the specified file address. You can suppress this check by setting the RCC on the level you specify to `'\0'`.

You test for both these errors by checking that the return value of the function is NULL.

Normal return

Pointer to the storage block containing the retrieved record image.

Error return

NULL if this, or any outstanding I/O operation was unsuccessful.

Detailed error information is contained in the error byte for data level *n* (`ecbptr()->ce1sud[n]`). See *ALCS Application Programming Guide*.

Loss of control

This function can cause the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return” above.

Example

This example reads a record from a general data set onto data level 7 (D7) and holds the file address. The program has already computed the file address and it is on data level 7 (D7).

ALCS returns control to the application program when the I/O is complete, and the record has been attached to the specified level.

```
#include <tpfeq.h>
#include <tpfio.h>

inm = fiwhc_ext(D7,FIND_GDS);

if (inm==NULL)
{
  errno=0x1234;
  perror("Error reading record on level D2");
  abort();
}
```

Related information

2.33, “findc — Read a DASD record” on page 86.
2.35, “find_record — Read a DASD record” on page 90.
2.37, “finhc — Read a DASD record and hold the file address” on page 97.
2.39, “finwc — Read a DASD record and wait for I/O completion” on page 101.
2.41, “fiwhc — Read a DASD record, hold the file address, and wait for I/O completion” on page 105.

2.43 flipc — Exchange the contents of two storage and data levels

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void flipc(enum t_lvl level1, enum t_lvl level2);
```

level1
One of the ECB levels (D0, D1, ..., DF) to be exchanged. See 2.1, “Common parameters” on page 25.

level2
The other ECB level (D0, D1, ..., DF) to be exchanged. See 2.1, “Common parameters” on page 25.

Description

Use the `flipc` function to exchange the contents of two storage levels and associated data levels (including data level extensions). `flipc` also exchanges the associated detail error bytes in the ECB.

Do not call `flipc` when either of the levels is currently specified in an I/O function that is still in progress. (That is, between the function call and the `waitc`, or a function containing an implied `waitc`, that follows it.)

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example exchanges the contents of levels 2 (D2) and 15 (DF).

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
flipc(D2,DF);
```

Related information

None.

2.44 gdsnc — Open or close a general data set

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int gdsnc(enum t_lvl level, unsigned char op, enum t_blktype size,
          int rrn_fmt, char *dsn);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

int gdsnc(TPF_DECB *decb, unsigned char op, enum t_blktype size,
          int rrn_fmt, char *dsn);
```

level
One of the ECB levels (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

op Specifies whether the data set is to be opened or closed. Use one of the values: GDSNC_OPEN, or GDSNC_CLOSE.

size
The size of the record. Use one of the defined block sizes: L1, L2, ..., L8.

rrn_fmt
This parameter is for compatibility with TPF and has no effect in ALCS. (In ALCS all general data sets are VSAM data sets.) Use one of the defined values: GDSNC_TPF_FMT or GDSNC_NOT_TPF_FMT.

dsn
A pointer to a 16-byte `char` array containing the data set name. The data set name must be left-justified and the array padded with blanks.

Description

Use the `gdsnc` function to open or close a general data set. (The action depends on the value of the *op* parameter.) When you use `gdsnc` to open a general data set, it returns the file address of a record in the general data set.

Before calling `gdsnc`, you must set up a value in the first fullword of the data extension field in the ECB level specified in the *level* parameter (`ce1fx`*n*), or in the DECB specified in the *decb* parameter (`idecfx0`). ALCS interprets this integer as the relative record number of the record you want to access. (The first record in a general data set has relative record number 0, the second 1, and so on.)

After calling `gdsnc` to open a general data set, an application program can read the record at the supplied file address (for example, by calling a `findc` function) or write a record to it (for example, by calling a `filec` function). However, the read or write function might require the application program to set up the record ID (and optionally the RCC) in the data level before calling the function. The `gdsnc` function does not set up these values.

When an application has read or written one record from the general data set, it can get the file address of subsequent records in the general data set by calling the `gdsrc` function.

When an entry has finished processing, it must close the general data set, using `gdsnc` with the GDSNC_CLOSE value of *op*, before exiting.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return
Zero.

Error return
ALCS sets this as follows:

4 Open completed without error but the relative record number is invalid (defined value BAD_DSN_RRN). The entry can use the general data set, but must first get a valid file address using `gdsrc`.
16 Open failed (defined value DSN_NOT_MNT). The entry cannot use the general data set.

Loss of control
This function does not cause the entry to lose control.

Examples
This example opens a general data set on ECB level 9 (D9). The data set name is VM1.KMV.FILE and it contains size L2 records.

Note: The data set name is padded with blanks.

```
#include <tpfeq.h>
#include <tpfapi.h>

int zero=0;

memcpy(&ecbptr->ce1fx9[0],&zero,4);
switch (gdsnc(D9,GDSNC_OPEN,L2,GDSNC_TPF_FMT,"VM1.KNV.FILE    "))
{
   case 0:  break;
   case 4:  errno = 0x123;
            perror("Relative record number is invalid");
            abort();
   case 16: errno = 0x234;
            perror("Cannot open the requested general data set");
            abort();
 }
```

This example opens a general data set and sets a relative record number to access the sixth record in that data set using the specified DECB. The data set name is VM1.KSC.FILE and it contains size L3 records.

Note: The data set name is padded with blanks.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB *decb;
int      rrn = 5;

memcpy(&decb->idecfx0,&rrn,4);
switch (gdsnc(decb,GDSNC_OPEN,L3,GDSNC_TPF_FMT,"VM1.KSC.FILE    "))
{
   case 0:  break;
   case 4:  errno = 0x123;
            perror("Relative record number is invalid");
            abort();
   case 16: errno = 0x234;
            perror("Cannot open the requested general data set");
            abort();
 }
```

Related information

2.45, “gdsrc — Compute the file address of a general data set record” on page 113.
2.59, “raisa — Compute the file address of a general file record” on page 144.

2.45 gdsrc — Compute the file address of a general data set record

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int gdsrc(enum t_lvl level, enum t_blktype size, char *dsn);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

int gdsrc(TPF_DECB *decb, enum t_blktype size, char *dsn);
```

level
One of the ECB levels (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

size
The size of the record. Use one of the defined block sizes: L1, L2, ..., L8.

dsn
A pointer to a 16-byte `char` array containing the data set name. The data set name must be left-justified and the array padded with blanks.

Description

Use the `gdsrc` function to compute the file address of a record in a general data set. The general data set must be open (opened using the `gdsnc` function.)

Before calling `gdsrc`, you must set up a value in the first fullword of the data extension field in the ECB level specified in the *level* parameter (`ce1fx`*n*) or in the DECB specified in the *decb* parameter (`idecfx0`). ALCS interprets this integer as the relative record number of the record you want to access. (The first record in a general data set has relative record number 0, the second relative record number 1, and so on.)

After calling `gdsrc` to calculate a general data set file address, an application program can read the record at this file address (for example, by calling a `findc` function) or write a record to it (for example, by calling a `filec` function). However, the read or write function might require the application program to set up the record ID (and optionally the RCC) in the data level before calling the function. The `gdsrc` function does not set up these values before calling the read or write function.

When the entry has finished processing, it must close the general data set, using `gdsnc`, before exiting.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return
Zero.

Error return
4 The general data set is open but the relative record number is invalid (defined value BAD_DSN_RRN).
16 The general data set is not open, so gdsrc cannot be used (defined value DSN_NOT_MNT). (Use the gdsnc function instead.)

Loss of control
This function does not cause the entry to lose control.

Examples
This example sets a relative record number to access the third record in a general data set using data level 3 (D3). The data set name is MAR.PNJ.FILE.

Note: The data set name is padded with blanks.

```
#include <tpfeq.h>
#include <tpfapi.h>

int rrn=2;

memcpy(&ecbptr->ce1fx3[0],&rrn,4);
switch (gdsrc(D9,L4,"MAR.PNJ.FILE    "))
{
   case 0:  break;
   case 4:  errno =0x123;
            perror("Relative record number is invalid");
            abort();
   case 16: errno =0x234;
            perror("General data set is not open");
            abort();
 }
```

This example sets a relative record number to access the second record in a general data set using the specified DECB. The data set name is JUN.PNJ.FILE.

Note: The data set name is padded with blanks.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB *decb;
int      rrn=1;

memcpy(&decb->idecfx0,&rrn,4);
switch (gdsrc(decb,L4,"JUN.PNJ.FILE    "))
{
   case 0:  break;
   case 4:  errno =0x123;
            perror("Relative record number is invalid");
            abort();
   case 16: errno =0x234;
            perror("General data set is not open");
            abort();
 }
```

Related information

2.44, “gdsnc — Open or close a general data set” on page 110.
2.59, “raisa — Compute the file address of a general file record” on page 144.

2.46 getcc — Get a storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void  *getcc(enum t_lvl level, enum t_getfmt format, third_parameter
            [,int fill_char]);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void  *getcc(TPF_DECB *decb, enum t_getfmt format, third_parameter
            [,int fill_char]);
```

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the ECB storage level on which you want the new block to be attached.

decb
A pointer to a data event control block (DECB). This is the DECB containing the storage level on which you want the new block to be attached.

format
This parameter specifies what the *third_parameter* in the function represents. Use one of the values:

GETCC_TYPE
The *third_parameter* is a block size code, L0, L1, ...L8, as defined by your installation.

GETCC_SIZE
The *third_parameter* is the size of the block, in bytes. ALCS allocates the smallest available block size that satisfies the request. Specify a number between 146 and the maximum block sizes defined in your installation.

GETCC_ATTR*n* or GETCC_PRIME or GETCC_OVERFLOW
The *third_parameter* is a pointer to a 2-character record ID. ALCS uses the block size defined for this record ID in the ALCS system generation.

When you specify GETCC_ATTR*n*, where *n* is a number 0 to 9, getcc uses this number *n* as the record ID qualifier. You can specify GETCC_PRIME as an alternative to GETCC_ATTR0 and GETCC_OVERFLOW as an alternative to GETCC_ATTR1.

You can additionally specify in the *format* parameter one or both of the following:

GETCC_FILL
Request getcc to fill the block with the character specified in the *fill_char* parameter. (By default, getcc fills the block with 0x00.)

TPF compatibility

The default storage block fill character in TPF varies by installation. On most TPF systems, the default is for TPF to fill the storage block with binary zeros as for ALCS. However, in some TPF installations the default is to leave the storage block uninitialized.

GETCC_COMMON
: This value is provided for compatibility with TPF. It has no effect in ALCS.

third_parameter
: The meaning of this parameter depends on the value you have specified in the *format* parameter, as described above.

fill_char
: Specify the character you want `getcc` to use to fill the storage block. Only use this parameter when you have specified GETCC_FILL in the *format* parameter.

Description

Use the `getcc` function to get and attach a storage block, of the requested size, to the ECB storage level you specify in the *level* parameter, or the storage level in the DECB you specify in the *decb* parameter. (There must be no storage block already attached at this level.)

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Pointer to the newly obtained storage block.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example gets storage blocks on ECB levels 2, 7, 9 and 15 (D2, D7, D9, and DF) and on a DECB. The first `getcc` call specifies the block size by coding the record ID, the second and third by specifying the block size code, and the fourth and fifth by specifying the size in bytes. The last 3 `getcc` calls fill the block with space characters. The last `getcc` call requests a storage block for a DECB.

Note: The first 4 `getcc` calls show examples of syntax accepted by the C compiler; the last `getcc` call shows an example of syntax accepted by the C++ compiler.

```
#include <tpfeq.h>
#include <tpfapi.h>
#include <c$am0sg.h>
#define BLANK (' ')

struct am0sg *amsg1;         /* pointers to message blocks  */
char         *work1, *work2, *work3, *work4;
TPF_DECB     *decb;
  ⋮
amsg1 = getcc(D2, GETCC_ATTR0, "OM");
work1 = getcc(D7, GETCC_TYPE, L2);   /* gets a size L2 block */
work2 = getcc(D9, GETCC_TYPE+GETCC_FILL, L2, BLANK);
work3 = getcc(DF, GETCC_SIZE+GETCC_FILL, 1100, BLANK);
work4 = (char *)getcc(decb1, (enum t_getfmt)(GETCC_SIZE+GETCC_FILL), 2000, BLANK);
```

Related information

2.53, “levtest — Test a storage level” on page 132.
2.60, “rcunc — Release a storage block and unhold a file address” on page 146.
2.61, “relcc — Release a storage block” on page 148.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.47 getfc — Get a pool-file record address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

unsigned int getfc(enum t_lvl level, int type, char *record_id, int block,
                   int error [,int fill_char]);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_FA8 getfc(TPF_DECB *decb, int type, char *record_id, int block,
                   int error [,int fill_char]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

type
Specify GETFC_TYPE*n*, where *n* is a number 0 to 9. `getfc` uses this number *n* as the record ID qualifier. You can specify GETFC_PRIME as an alternative to GETFC_TYPE0 and GETFC_OVERFLOW as an alternative to GETFC_TYPE1.

record_id
Pointer to a 2-character string which contains the record ID of the pool-file record that you want.

block
Specifies whether or not you want to get a storage block at the same time as a pool-file record. (The storage block has the same size as the pool-file record.) Code GETFC_BLOCK to get a block and a file address, or GETFC_NOBLOCK to get only a file address. (When you code GETFC_BLOCK in the *block* parameter, the specified ECB storage level or DECB must not have an attached storage block.)

By default, ALCS sets the contents of a new storage block to binary zeros. However, you can optionally also code GETFC_FILL in addition to GETFC_BLOCK. (Separate the two values with a bitwise or (|) symbol.) If you do so, ALCS fills the storage block with the symbol you supply in the *fill_char* character.

TPF compatibility

The default storage block fill character in TPF varies by installation. On most TPF systems, the default is for TPF to fill the storage block with binary zeros as for ALCS. However, in some TPF installations the default is to leave the storage block uninitialized.

Note: When you code GETFC_BLOCK, you can additionally code GETFC_COMM or GETFC_NOCOMM in the *block* parameter. This is for compatibility with TPF. ALCS ignores either value.

error
Specifies whether you want ALCS to return control to the program when an error occurs. Code GETFC_SERRC to transfer control to the system error routine (with exit) when the ALCS cannot get a file address or storage block. Code GETFC_NOSERRC if you want control returned to the program.

fill_char
Specify the character you want `getfc` to use to fill the storage block. Only use this parameter when you have specified GETFC_BLOCK and GETFC_FILL in the *block* parameter.

Description

Use the `getfc` function to get a pool file address (and optionally, a working storage block) for use by the program.

The *record_id* and *type* parameters determine the type of pool-file record (long-term or short-term pool) and its block size (L1, L2, L3, ... L8).

In the ALCS generation, the system programmer allocates record IDs so that ALCS can determine the type and size of a requested pool-file record from the record ID and qualifier.

Type `TPF_FA8` is defined in <c$decb.h> (#included by <tpfio.h>).

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Unsigned integer value representing a file address, or an 8-byte file address in 4x4 format (defined type `TPF_FA8`).

Error return

Integer value of zero or type `TPF_FA8` with a value of zero, when GETFC_NOSERRC is coded, otherwise not applicable.

Loss of control

This function can cause the entry to lose control.

Examples

1. This example gets a file address for a message block on ECB level 2 (D2), and a storage block of the appropriate size on the same level. The record ID is `'OM'` with a qualifier of 0.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$am0sg.h>

struct am0sg *amsg               /* pointers to message blocks       */
  ⋮
amsg = ecbptr()->ce1cr1;         /* Base prime message block         */
if (!(amsg->am0fch = getfc(D2,GETFC_PRIME,"OM",GETFC_BLOCK,GETFC_NOSERRC)))
  exit(0x33001);                 /* Dump with exit if getfc() failed */
```

2. This example gets a file address on ECB level 15 (DF) together with a storage block of the appropriate size on the same level. The record ID is `'XI'` with a

qualifier of 1. It fills the storage block with the character 'F' and prints the file address.

```
#include <tpfeq.h>
#include <tpfio.h>
unsigned int fa;

fa = getfc(DF,GETFC_TYPE1,"XI",GETFC_BLOCK|GETFC_NOCOMM|
          GETFC_FILL,GETFC_NOSERRC,'F');
printf("file address obtained = %08X",fa);
```

3. This example gets a file address for a message block on a DECB together with a storage block of the appropriate size on the same DECB. The record ID is 'OM' with a qualifier of 0.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$am0sg.h>
TPF_FA8  fa;
TPF_DECB *decb;

if (!(fa = getfc(decb,GETFC_TYPE1,"OM",(int)(GETFC_BLOCK|GETFC_FILL),
          GETFC_NOSERRC,' ')))
  exit(0x33001);                /* Dump with exit if getfc() failed */
```

Related information

2.48, “getfc_alt — Get a pool-file record address” on page 122.
2.62, “relfc — Release a pool-file record address” on page 150.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.48 getfc_alt — Get a pool-file record address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

unsigned int getfc_alt(enum t_lvl level, int term, enum t_blktype size, int block, int error);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_FA8 getfc_alt(TPF_DECB *decb, int term, enum t_blktype size, int block, int error);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

term
Specify GETFC_LONG for long-term pool or GETFC_SHORT for short-term pool.

size
A block size code (L0, L1 ..., L8) as defined by your installation.

block
Specifies whether or not you want to get a storage block at the same time as a pool-file record. (The storage block has the same size as the pool-file record.) Code GETFC_BLOCK to get a block and a file address, or GETFC_NOBLOCK to get only a file address. (When you code GETFC_BLOCK in the *block* parameter, the specified ECB storage level or DECB must not have an attached storage block.)

error
Specifies whether you want ALCS to return control to the program when an error occurs. Code GETFC_SERRC to transfer control to the system error routine (with exit) when ALCS cannot get a file address or storage block. Code GETFC_NOSERRC if you want control returned to the program.

Description

Use the getfc_alt function to get a pool file address (and optionally, a working storage block) for use by the program without specifying a record_id. The type of pool-file record (long-term or short-term) and its block size (L1, L2, L3, ... L8) are explicitly coded.

Type TPF_FA8 is defined in <c$decb.h> (#included by <tpfio.h>).

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Unsigned integer value representing a file address, or an 8-byte file address in 4x4 format (defined type TPF_FA8).

Error return

Integer value of zero or type TPF_FA8 with a value of zero, when GETFC_NOSERRC is coded, otherwise not applicable.

Loss of control

This function can cause the entry to lose control.

Examples

This example gets an L3 long-term pool file address on ECB level 15 (DF) together with a storage block of the same size on the same level.

```
#include <tpfeq.h>
#include <tpfio.h>
unsigned int fa;

fa = getfc_alt(DF,GETFC_LONG,L3,GETFC_BLOCK,GETFC_NOSERRC);
```

This example gets an L2 short-term pool file address on the level of a DECB together with a storage block of the same size on the same level.

```
#include <tpfeq.h>
#include <tpfio.h>
TPF_FA8  fa;
TPF_DECB *decb;

fa = getfc_alt(decb,GETFC_SHORT,L2,GETFC_BLOCK,GETFC_NOSERRC);
```

Related information

2.47, “getfc — Get a pool-file record address” on page 119.
2.62, “relfc — Release a pool-file record address” on page 150.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.49 glob — Address an application global field or record

Format

```
#include <tpfeq.h>
#include <tpfglbl.h>

void *glob(char *tagname);
```

tagname
This argument, defined in the header file <c$globz.h> (#included by <tpfglbl.h>), identifies the global field or record. (Your system programmer should be able to tell you the names used for global fields and records.)

Description

Use the glob function to return a pointer to a global field or record defined in <c$globz.h>. If you use a global field or record name not defined in <c$globz.h>, this will give unpredictable results.

This function provides read only reference to the specified global. To modify global fields, use the global function.

Normal return

Pointer which references byte 0 of the specified global field or record.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example displays the contents of the global field _HAALC.

```
#include <tpfeq.h>
#include <tpfglbl.h>

printf("The host airline code is: %2.2s/n",glob(_HAALC));
```

Related information

1.15, “<c$globz.h>” on page 14.
2.50, “global — Operate on an application global field” on page 125.

2.50 global — Operate on an application global field

Format

```
#include <tpfeq.h>
#include <tpfglbl.h>

void global(unsigned int tagname, enum t_glbl action,
void *pointer);
```

tagname
This argument, defined in the header file <c$globz.h> (#included by <tpfglbl.h>), identifies the global field. (Your system programmer should be able to tell you the names used for global fields.)

action
The action you want to be performed against *tagname*. Use one of the following terms:

GLOBAL_MODIFY
ALCS replaces the value of the specified global field with the value specified by the parameter *pointer*.

ALCS does not perform any additional actions on the global field (keypointing, processor synchronization, and so on). To perform any of these actions, code calls to global specifying other parameters described below.

GLOBAL_KEYPOINT
ALCS keypoints the record containing the specified global field.

You must code a NULL value for the *pointer* parameter.

GLOBAL_LOCK
Reserve the exclusive use of the specified global field. The contents of the specified global field are copied to user storage (pointed to by parameter *pointer*) for the defined length of the global field.

Note: To minimize performance problems resulting from other entries awaiting locks, ensure that a lock is held for the shortest possible time.

GLOBAL_UNLOCK
ALCS removes the entry's exclusive use of the specified global field. The entry must have previously performed a GLOBAL_LOCK on the specified global field before using GLOBAL_UNLOCK.

You must code a NULL value for the *pointer* parameter.

GLOBAL_COPY
Copies the contents of the specified global field to user storage (pointed to by parameter *pointer*) for the defined length of the global field.

GLOBAL_SYNC
This informs ALCS that the field has changed. If the record that contains the field is keypointable, ALCS updates the file copy of the record.

In a loosely-coupled configuration, this service starts the process of updating the field in other processors in the complex.

ALCS removes the entry's exclusive use of the specified global field. The entry must have previously performed a GLOBAL_LOCK on the specified global field before using GLOBAL_SYNC.

You must code a NULL value for the *pointer* parameter.

GLOBAL_UPDATE
ALCS replaces the value of the specified global field with the value specified by the parameter *pointer*.

ALCS then proceeds as with the GLOBAL_SYNC action. If the record containing the field is keypointable, ALCS updates the file copy of the record.

In a loosely-coupled configuration, this service starts the process of updating the field in other processors in the complex.

ALCS removes the entry's exclusive use of the specified global field. The entry must have previously performed a GLOBAL_LOCK on the specified global field before using GLOBAL_UPDATE.

You must code a NULL value for the *pointer* parameter.

pointer
A pointer to an area, the contents of which depend on the value of the *action* parameter, as described above. It can be either a source or destination field, depending on the value of *action*, or a NULL.

The length of the field is the length of the global field, provided in the *tagname* definition.

You must supply this parameter when you specify GLOBAL_MODIFY, GLOBAL_LOCK, GLOBAL_COPY, or GLOBAL_UPDATE as values for the *action* parameter. With other values of *action*, you must code a NULL value for the *pointer* parameter.

Description

Use the `global` function to address, update, synchronize, or keypoint a global field.

You can only perform operations on global fields with `global` (not on global records). When *tagname* is a record and not a field, ALCS does nothing.

Programs using the GLOBAL_LOCK option must issue a GLOBAL_UNLOCK as soon as possible, in order to prevent severe performance degradation. This is because other entries may be awaiting locks on global fields in the target global block.

Programs using the GLOBAL_LOCK, GLOBAL_UNLOCK, or GLOBAL_SYNC options should not have any pending I/O operations outstanding.

The GLOBAL_UPDATE value for *action* is provided for TPF compatibility.

The recommended sequence of actions for updating global fields is:

GLOBAL_LOCK
GLOBAL_MODIFY
GLOBAL_SYNC

Normal return
Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Example
This example updates, keypoints, and synchronizes the global field _NS1NS.

```
#include <tpfeq.h>
#include <tpfglbl.h>
  ⋮
int  ns1ns;
  ⋮
waitc();       /* wait for any outstanding I/O to complete */
  ⋮
global (_NS1NS,GLOBAL_LOCK,&ns1ns);     /* lock the global and copy  */
++ns1ns;                                /* update the data           */
global (_NS1NS,GLOBAL_MODIFY,&ns1ns);   /* modify the global data    */
global (_NS1NS,GLOBAL_SYNC,NULL);       /* synchronize globals       */
```

Related information
1.15, “<c$globz.h>” on page 14.
2.49, “glob — Address an application global field or record” on page 124.

2.51 helpc — Provide context sensitive help

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void helpc(int function, char *ptopic, char *stopic);
```

Where:

function
The action to be performed. Specify one of:

HELPC_SET_CONTEXT
Set the default primary and optional secondary help topic for the ZHELP command for this end user.

This default remains in effect until either:
- A subsequent HELPC macro or helpc function changes the default, or:
- The end user enters another input message or command.

HELP_EXTRACT_CONTEXT
Retrieve the default primary and optional secondary help topic for the ZHELP command for this end user.

ptopic
For HELPC_SET_CONTEXT, this is a pointer to the primary topic text string. The string must not exceed 8 characters and must be padded as required with space (X'40') characters. It must be in upper case and cannot include spaces or special characters other than:

Star or asterisk (*)
Commercial at (@)
Hyphen or minus (–)
Hash or pound (#)

When you want to clear the primary context, code NULL.

For HELPC_EXTRACT_CONTEXT, this is a pointer to the area where the primary text string should be stored.

stopic
For HELPC_SET_CONTEXT, this is a pointer to the secondary topic text string. The string must not exceed 8 characters and must be padded as required with space (X'40') characters. It must be in upper case and cannot include spaces or special characters other than:

Star or asterisk (*)
Commercial at (@)
Hyphen or minus (–)
Hash or pound (#)

When you want to clear the secondary context, code NULL.

For HELPC_EXTRACT_CONTEXT, this is a pointer to the area where the secondary text string should be stored.

Description

Use the `helpc` function to set or retrieve the default primary and optional secondary help topic for the `ZHELP` command for this end user.

Normal return

For HELPC_EXTRACT_CONTEXT, there is a pointer to the primary and optional secondary help topics.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example sets the primary help topic to "HELP " and clears the secondary help topic. Subsequently, the string "HELP " is stored at EBW000. Since the secondary help topic has been cleared, 8 space (X'40') characters are stored at EBW008.

```
#include <tpfeq.h>
#include <tpfapi.h>

char *ptopic;
char *stopic;
    .
    .
    .
ptopic = "HELP    ";
helpc (HELPC_SET_CONTEXT,ptopic,NULL);
    .
    .
helpc (HELPC_EXTRACT_CONTEXT,&ecbptr()->ebw000,&ecbptr()->ebw008);
```

Related information

ALCS Application Programming Guide

2.52 IPRSE_parse — Parser utility

TPF compatibility

ALCS supports the IPRSE_parse function to assist in porting applications from TPF. IPRSE_parse is described in *TPF C/C++ Language Support User's Guide*. It is not an intended programming interface of ALCS. Do not use IPRSE_parse in applications you are developing for ALCS.

Note: ALCS does not support the TPF IPRSE_bldprstr function.

Format

```
#include <tpfparse.h>

int IPRSE_parse(char *string, const char *grammar,
                struct IPRSE_output *result, int options,
                const char *errheader);
```

string
Pointer to the text you want to parse; a string of up to 600 characters. The string must be terminated with a NULL unless the IPRSE_EOM option (see below) is specified.

grammar
Pointer to the grammar for parsing the text; a NULL-terminated string of up to 1200 characters. Please refer to *TPF C/C++ Language Support User's Guide* for an explanation of this string.

result
Pointer to structure IPRSE_output (defined in <tpfparse.h>) which contains information about the first parameter in the text. The IPRSE_output structure contains the following fields:

IPRSE_parameter
Pointer to the parameter name.

IPRSE_value
Pointer to the value of the parameter.

IPRSE_next
Pointer to structure IPRSE_output which contains information about the next parameter in the text.

options
Options for parsing the text. Use the sum of one or more of the following defined values:

IPRSE_ALLOC
This option is mandatory for ALCS.

IPRSE_PRINT
IPRSE_parse builds and sends a standard error message to the originating terminal if it detects a syntax error in the text.

IPRSE_NOPRINT
IPRSE_parse does not send any error response message to the originating terminal.

IPRSE_EOM
The text is terminated with an end-of-message character (_EOM) instead of a NULL. `IPRSE_parse` replaces the _EOM with a NULL before parsing the text.

errheader
Pointer to a 4-character string which `IPRSE_parse` uses as the message prefix if it sends an error response to the originating terminal.

Description
Please refer to *TPF C/C++ Language Support User's Guide* for a description of this function.

Normal return
The function returns a positive integer which is the number of parameters in the text.

Error return
One of the following defined values:

IPRSE_HELP The text is '?' or 'HELP'

IPRSE_BAD The text does not comply with the grammar.

Loss of control
This function can cause the entry to lose control.

Example
No example provided. IBM recommends you not to use this function in ALCS.

Related information
TPF C/C++ Language Support User's Guide

2.53 levtest — Test a storage level

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int levtest(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

int levtest(TPF_DECB *decb);
```

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

Description

Use the levtest function to test whether a storage block is attached at the specified ECB storage level or DECB.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

When the ECB level or DECB has an attached storage block, this is an integer value representing the size of the storage block in bytes. When the ECB level or DECB does not have an attached storage block, ALCS returns a zero value.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example gets and addresses a storage block on ECB level 12 (DC) if it does not already have an attached storage block. (If level 12 (DC) has an attached block, the example addresses this existing block.)

```
#include <tpfeq.h>
#include <tpfapi.h>

char *work;
  ⋮
if(!levtest(DC))
 work = getcc(DC, GETCC_TYPE, L2);
else
 work = ecbptr()->ce1crc;
```

This example gets and addresses a storage block on the level of a DECB if it does not already have an attached storage block.

```
#include <tpfeq.h>
#include <tpfapi.h>

char     *work;
TPF_DECB *decb;
  ⋮
if(!levtest(decb))
 work = (char *)getcc(decb, GETCC_TYPE, L1);
```

Related information

2.14, “crusa — Release storage blocks from specified levels” on page 49.
2.46, “getcc — Get a storage block” on page 116.
2.60, “rcunc — Release a storage block and unhold a file address” on page 146.
2.61, “relcc — Release a storage block” on page 148.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.54 lodic — Get system load information

TPF Compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int lodic(void);
```

Description

Use the lodic function to calculate the number of additional entries that the current entry can create without performance degradation. lodic is designed for use by applications which create several entries to do simultaneous processing. The entry is marked as a maintenance entry.

You are recommended to call lodic before creating each new entry.

Normal return

The maximum number of additional entries that the entry can create.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example creates up to 10 new entries. However, it only creates the new entries provided that lodic says that another 5 new entries would not degrade performance.

```
#include <tpfeq.h>
#include <tpfapi.h>

int entries = 0;

/* create up to 10 entries but                    */
/* ensure system has spare capacity for 5 entries */

while ( entries < 10 && lodic()>5 )
{
  cremc(3,"XYZ",ABC1);                /* create entry */
  ++entries;
}
defrc();          /* defer processing until able to create more entries */
```

Related information

2.9, “credc — Create an entry for deferred scheduling” on page 39.
2.10, “creec — Create an entry with an attached storage block” on page 41.
2.11, “cremc — Create an entry for immediate scheduling” on page 43.

2.55 lodic_ext — Get system load information and mark ECB with extended options

TPF Compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int lodic_ext(unsigned int opt, void *usprm);
```

Description

Use the lodic_ext function to calculate the number of additional entries that the current entry can create without performance degradation and to determine if an ECB can be suspended (based on the level of available resources). lodic_ext is designed for use by applications which create several entries to do simultaneous processing.

opt

The *opt* flag specifies options that indicate ECB behavior. The following values for *opt* are valid:

- **LODIC_ECBCREATE**
  Flag the ECB as a maintenance ECB and return the maximum number of entries that the current entry can create without performance degradation. lodic_ext is designed for use by applications which create several entries to do simultaneous processing. This option functions identically to the lodic() api. You are recommended to call lodic or lodic_ext before creating each new entry.

  Normal return
  The maximum number of additional entries that the entry can create. A value of zero means no more entries can be created without performance degradation.

  Error return
  Not applicable.

  Loss of control
  This function does not cause the entry to lose control.

- **LODIC_CHECK**
  As LODIC_ECBCREATE but does not flag the ECB as a maintenance ECB.

- **LODIC_SUSPEND**
  Causes the ECB to lose control if no additional entries can be accepted without performance degradation. Control is returned when one or more entries can be accepted. Flag the ECB as a maintenance ECB.

  Normal return
  No value is returned

  Error return
  Not applicable.

Loss of control
This function may cause the entry to lose control.

- **LODIC_UNMARK**
  Unmark the ECB as a maintenance ECB (reset the flag)

  Normal return
  No value is returned

  Error return
  Not applicable.

  Loss of control
  This function does not cause the entry to lose control.

Product-Sensitive Programming Interface

TPF Compatibility

Do not use this function in programs that must be compatible with TPF.

- **LODIC_DACV**
  Return a pointer to the Activity Control Variables . See c$cuacv.h for details

  Normal return
  Pointer to the Activity Control Variables

  Error return
  Not applicable.

  Loss of control
  This function does not cause the entry to lose control

End of Product-Sensitive Programming Interface

Product-Sensitive Programming Interface

TPF Compatibility

Do not use this function in programs that must be compatible with TPF.

- **LODIC_CONDITIONAL_DEFER**

  Force the entry to lose control if and only if the entry needs to lose control to avoid locking out other entries.

  Normal return
  No value is returned

  Error return
  Not applicable.

  Loss of control
  This function may cause the entry to lose control

End of Product-Sensitive Programming Interface

- **LODIC_HOLD**
  This parameter is included for TPF compatibility but has no effect on the api
- **LODIC_BATCH**, **LODIC_LOBATCH**, **LODIC_IBMHI**, **LODIC_IBMLO**
  These parameters are included for TPF compatibility but have no effect on the api. Lodic classes are not supported in ALCS.

usprm
This parameter is included for TPF compatibility but has no effect on the application programming interface (API).

Programming notes

The `lodic_ext` API is an extension of the `LODIC` API and provides interface compatibility with the TPF version of &Lodicxt.. The return values are different in some cases and some TPF options are not supported.

See the 2.54, "`lodic` — Get system load information" on page 134 for more details on how to use the `lodic_ext` API . Note that each `lodic_ext` function calls the equivalent assembler `LODIC` macro to execute the API.

Example

This example creates up to 10 new entries. However, it only creates the new entries provided that the results from a `lodic_ext` call say that another 5 new entries would not degrade performance.

```
#include <tpfeq.h>
#include <tpfapi.h>
int entries = _;
/* create up to 1_ entries but */
/* ensure system has spare capacity for 5 entries */
while ( entries < 1_ && lodic_ext(LODIC_ECBCREATE)>5 )
{
 cremc(3,"XYZ",ABC1); /* create entry */
 ++entries;
}
defrc(); /* defer processing until able to create more entries */
```

Related information

2.9, "`credc` — Create an entry for deferred scheduling" on page 39.
2.10, "`creec` — Create an entry with an attached storage block" on page 41.
2.11, "`cremc` — Create an entry for immediate scheduling" on page 43.
2.54, "`lodic` — Get system load information" on page 134.

2.56 longc — Set maximum entry life

TPF compatibility

ALCS supports this function for compatibility with TPF. If your program does not need to be compatible with TPF, use the slimc function instead.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

short int longc(unsigned short int longc_req);
```

longc_req
One of the following, specifying the lifetime for the entry:

LONGC_INDEF
Remove the entry life limit. In ALCS, this sets the entry life to the maximum system limit.

LONGC_CLEAR
Add one minute to the current entry life.

n Additional entry life in minutes (note: use an integer in the range 1 through 254).

Description

You can use the longc function to adjust the maximum lifetime of the entry.

Normal return

Short integer value of zero.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example sets the life of the current entry to be indefinitely long.

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
longc(LONGC_INDEF);          /* Set entry life to be indefinitely long */
```

Related information

2.69, "slimc — Set or remove processing limits for the entry" on page 168.

2.57 modec — Set current addressing mode

TPF compatibility

ALCS supports this function for compatibility with some versions of TPF. It has no effect in ALCS C programs. IBM recommends you not to use this function in ALCS.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void modec(enum t_modec mode);
```

mode
The number of bits to be used for main storage addressing. Use the values MODEC_24 for 24-bit addressing mode, and MODEC_31 for 31-bit addressing.

Description

In some versions of TPF you can use the `modec` function to set the current addressing mode to 24-bit or 31-bit. In ALCS, `modec` has no effect.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

No example provided. IBM recommends you not to use this function in ALCS.

Related information

None.

2.58 mqawait — Wait for one or more asynchronous MQI calls to complete

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void mqawait(void);
```

Description

Use the mqawait function to wait for one or more outstanding asynchronous MQI calls to complete.

The mqawait function waits only for asynchronous MQI calls to complete and does not wait for the completion of any other ALCS I/O operations. The I/O counter and error indicators in the ALCS ECB are not affected.

Note: MQGET has an option to allow the operating system to notify (or **signal**) a program when an expected message arrives on a queue. This allows the program to continue with other work. *WebSphere MQ for z/OS Application Programming Reference* explains how to do this. IBM recommends that you use this method instead of waiting for messages.

Normal return

Void. At least one of the asynchronous MQI functions issued by this ECB has completed. You must determine the status of each outstanding asynchronous MQI function by testing the post bit in the first byte of the event control block. You can determine the status of each function by testing the completion code in the event control block.

To set a signal, use the MQGMO_SET_SIGNAL option in the Options field of the MQGMO structure when you use the MQGET call.

Error Return

Not Applicable.

Loss of Control

This function can cause the entry to lose control.

Example

The following example shows the use of mqawait in C. The sample function (or subroutine) gets a message from an MQI queue. If the queue is empty it waits up to 20 seconds for a message to arrive.

```
#pragma strings (readonly)
#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfio.h>
#include <string.h>
#include <cmqc.h>

/*-------------------------------------------------*/
/*                                                 */
/*  mqiget - sample routine to get a message       */
/*           from a MQM queue using signal         */
/*           request and mqawait()                 */
/*                                                 */
/*           input parameter is the name of the    */
/*           queue.                                */
/*                                                 */
/*           returns a pointer to the received     */
/*           message text or NULL if no message    */
/*           was received or an error occured.     */
/*-------------------------------------------------*/
void* mqiget(char* queue)
  {
    static MQBYTE  Buffer[2048] = "";     /* buffer for input */
    MQBYTE *RC = NULL;                    /* return code      */

    MQHCONN    Hconn = 0;
    MQOD       ObjDesc = { MQOD_DEFAULT };
    MQLONG     Options = 0;
    MQHOBJ     Hobj;
    MQLONG     CompCode;
    MQLONG     Reason;
    MQMD       MsgDesc = { MQMD_DEFAULT};
    MQGMO      GetMsgOpts = { MQGMO_DEFAULT};
    MQLONG     BufferLength = sizeof(Buffer);
    MQLONG     DataLength;
    MQLONG     EVCB;      /* Event Control Block for wait */


    /* macro to extract the user completion code from EVCB */
    #define    UCC(x) (x & 0x000000FF)

    memset(Buffer,0,BufferLength);   /* clear the input buffer */


    /* open the queue */

    Options =  MQOO_INPUT_SHARED;
    strcpy(ObjDesc.ObjectName , queue);
    ObjDesc.ObjectType = MQOT_Q;

    MQOPEN ( Hconn,
             &ObjDesc,
             Options,
             &Hobj,
             &CompCode,
             &Reason
           );
```

```
if (CompCode != MQCC_OK)
{
  printf("MQOPEN failed\n");
  printf("CompCode=%08X Reason=%08X\n",CompCode,Reason);
}
else
{
  GetMsgOpts.Options = MQGMO_SET_SIGNAL;
  GetMsgOpts.WaitInterval = 20000;    /* wait upto 20 seconds  */
  GetMsgOpts.Signal1  = &EVCB;        /* set pointer to EVCB   */
  EVCB = 0;                           /* clear the EVCB        */

  MQGET  ( Hconn,
           Hobj,
           &MsgDesc,
           &GetMsgOpts,
           BufferLength,
           &Buffer,
           &DataLength,
           &CompCode,
           &Reason
         );

  if (CompCode == MQCC_FAILED)
  {
    printf("MQGET with signal request not accepted\n");
    printf("CompCode=%08X Reason=%08X\n",CompCode,Reason);
  }

  if (CompCode == MQCC_WARNING &&
      Reason == MQRC_SIGNAL_REQUEST_ACCEPTED)
  {
    slimc(SLIMC_ECBLIFE,1,SLIMC_MINUTES);

    mqawait();                    /* wait for message */

    printf("mqawait completed - EVCB = %08X\n",EVCB);


    if (UCC(EVCB) == MQEC_WAIT_INTERVAL_EXPIRED)
      printf("wait interval expired\n");

    if (UCC(EVCB) == MQEC_WAIT_CANCELED)
      printf("wait canceled\n");

    if (UCC(EVCB) == MQEC_MSG_ARRIVED)
    {
      printf("message arrived\n");

      GetMsgOpts.Options = MQGMO_NO_WAIT;
```

```
        MQGET  ( Hconn,
                 Hobj,
                 &MsgDesc,
                 &GetMsgOpts,
                 BufferLength,
                 &Buffer,
                 &DataLength,
                 &CompCode,
                 &Reason
               );

        if (CompCode != MQCC_OK)
        {
          printf("MQGET for arrived message failed\n");
          printf("CompCode=%08X Reason=%08X\n",CompCode,Reason);
        }
      }
    }


    if (CompCode == MQCC_OK)
    {
      printf("MQGET for message succeeded\n");
      printf("message is:\n%-*s\n",DataLength,Buffer);
      RC = Buffer;
    }

    Options =  MQCO_NONE;

    MQCLOSE( Hconn,
             &Hobj,
             Options,
             &CompCode,
             &Reason
           );
    if (CompCode != MQCC_OK)
    {
      printf("MQCLOSE failed\n");
      printf("CompCode=%08X Reason=%08X\n",CompCode,Reason);
    }
  }
  return RC;
}
```

Related information

ALCS Application Programming Guide.
WebSphere MQ for z/OS Application Programming Reference.
2.2, “`atawait` — Wait for one or more asynchronous APPC/MVS calls to complete” on page 27.

2.59 raisa — Compute the file address of a general file record

TPF compatibility

This function is not source-compatible with the raisa function in TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void raisa(int type, int count, enum t_lvl level);
```

type

Use one of the following values:

RAISA_FIRST
: The function returns the file address of the first record (record number zero) in the general file specified in the *count* parameter.

RAISA_INC
: The function treats the *count* parameter as an **increment**. ALCS returns the file address of the (current record plus the increment).

count

When you code *type* as RAISA_FIRST, *count* must contain a general file number (1 through 255) which identifies the general file.

When you code *type* as RAISA_INC, *count* must contain a positive (increment) value. ALCS takes the file address of the general file record whose file address is currently on the data level specified in the *level* parameter and regards this general file record as the "current record" and notes its relative record number from the start of the file. ALCS then adds the increment to this current record number to calculate the relative record number of the record you require. For example, if the current record is relative record number 7 and *count* has a value of 10, the function returns the file address of relative record number 17.

level

A data level (D0, D1, ..., DF), see 2.1, "Common parameters" on page 25.

Description

Use the raisa function to return the file address of a general file record.

Normal return

Void. ALCS puts the file address of the general file record in the specified data level.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example reads 10 records, starting at the first, from general file number 7.

```
#include <tpfeq.h>
#include <tpfapi.h>
#define dump_no 1234

int i;
/* read 10 records from general file 7 */
for (i=0; i<10; ++i)
{
  if (i==0)
    raisa(RAISA_FIRST,7,D3);     /* get address of first record    */
  else
    raisa(RAISA_INC,1,D3);       /* get address of next record     */
  if (!finwc(D3)) exit(dump_no)  /* get the record - exit on error */
  ...                            /* process the record             */
  relcc(D3);                     /* free storage block             */
}
```

Related information

2.44, “gdsnc — Open or close a general data set” on page 110.
2.45, “gdsrc — Compute the file address of a general data set record” on page 113.

2.60 rcunc — Release a storage block and unhold a file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void rcunc(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void rcunc(TPF_DECB *decb);
```

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. Identifies the ECB level containing the address of the storage block to be released and the file address to be unheld.

decb
A pointer to a data event control block (DECB). Identifies the DECB containing the address of the storage block to be released and the file address to be unheld.

Description

Use the rcunc function to release a storage block and unhold a file address.

rcunc is the same as relcc followed by unfrc_ext or unfrc_ext. See the **Description** sections for relcc , unfrc and unfrc_ext.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example releases a storage block and unholds the record on ECB level 2.

```
#include <tpfeq.h>
#include <tpfio.h>
  ⋮
rcunc(D2)
```

This example releases a storage block and unholds the record on a DECB.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_DECB *decb;
  ⋮
rcunc(decb);
```

Related information

2.46, “getcc — Get a storage block” on page 116.
2.61, “relcc — Release a storage block” on page 148.
2.95, “unfrc — Unhold a file address” on page 218.
2.96, “unfrc_ext — Unhold a file address, with extended options” on page 220.

2.61 relcc — Release a storage block

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void relcc(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

void relcc(TPF_DECB *decb);
```

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. ALCS releases the block currently attached at this level.

decb
A pointer to a data event control block (DECB). ALCS releases the block currently attached to this DECB.

Description

Use the relcc function to release the storage block attached at the ECB storage level you specify in the *level* parameter, or the storage level of the DECB you specify in the *decb* parameter.

Ensure that there is a storage block attached at the specified ECB level or DECB when you call relcc.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example tests for the presence of a storage block on ECB level 11 (DB) and releases it if it exists.

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
if (levtest(DB))
   relcc(DB);
```

This example tests for the presence of a storage block on attached to a DECB and releases it if it exists.

```
#include <tpfeq.h>
#include <tpfapi.h>

TPF_DECB *decb;
  ⋮
if (levtest(decb))
   relcc(decb);
```

Related information

2.14, “crusa — Release storage blocks from specified levels” on page 49.
2.46, “getcc — Get a storage block” on page 116.
2.60, “rcunc — Release a storage block and unhold a file address” on page 146.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.62 relfc — Release a pool-file record address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void relfc(enum t_lvl level);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void relfc(TPF_DECB *decb);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the ECB data level containing the pool-file record address that is to be released.

decb
A pointer to a data event control block (DECB). This is the DECB containing the pool-file record address that is to be released.

Description

Use the relfc function to release the pool-file record address currently in the ECB data level that you specify in the *level* parameter, or the data level of the DECB that you specify in the *decb* parameter.

If the file address you are releasing exists in other records, or in the application global area, set these references to zero before calling relfc. Otherwise the database will be inconsistent.

Do not reference a file address after it has been released.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Examples

The following example releases all forward chain pool-file records from a message block, and sets the forward chain field in the prime message block to zero. The prime message block is attached at level 1 (D1).

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$am0sg.h>

struct am0sg *prime,*chain;          /* Pointers to message blocks  */
unsigned int adrs[100];
int j,i = 0;

prime = ecbptr()->ce1cr1;            /* Base prime message block    */
chain = prime;

while(chain->am0fch != 0)            /* Obtain all chain addresses  */
   {
    adrs[i] = chain->am0fch;
    crusa(D2);                       /* Release last storage block  */
    chain = find_record(D2,&adrs[i++],"OM",'\0',NOHOLD);
   }

for(j = 0; j < i; j++)               /* Release all chain addresses */
   {
    ecbptr()->ce1fm2 = adrs[j];
                                     /* Set up data level           */
    relfc(D2);                       /* and release the address     */
   }
```

The following example releases a pool-file record that is held in the DECB named POOLREC.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$decb.h>

TPF_DECB *decb;
DECBC_RC rc;
char     decb_name[17] = "POOLREC         ";
  ⋮
if ((decb = tpf_decb_locate(decb_name, &rc)) != NULL)
  relfc(decb);
else
{
  /error /
}
```

Related information

2.47, “getfc — Get a pool-file record address” on page 119.
2.48, “getfc_alt — Get a pool-file record address” on page 122.
2.63, “rlcha — Release a chain of pool-file record addresses” on page 152.
2.90, “tpf_rcrfc — Convert a file address” on page 209.

2.63 rlcha — Release a chain of pool-file record addresses

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void rlcha(struct stdhdr *header);
```

header
Pointer to a standard header in a prime record. This header contains a file address of a pool-file record, which can be the start of a chain of other pool-file records.

Description

Use the rlcha function to release the file addresses of all pool-file records chained from the specified record header in a prime record. ALCS assumes standard forward chaining from the prime record.

All the records in the chain must have the same record ID and record code check (RCC) as the first record in the chain.

Do not reference any of the released file addresses after they have been released.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example releases the chain of pool-file record addresses beginning with the address found in the standard header of the record in the storage block attached at storage level 5 (D5).

```
#include <tpfeq.h>
#include <tpfapi.h>

struct stdhdr *cp0hdr;
  ⋮
cp0hdr = ecbptr()->ce1cr5;
rlcha(cp0hdr);
```

Related information

2.47, “getfc — Get a pool-file record address” on page 119.
2.48, “getfc_alt — Get a pool-file record address” on page 122.
2.62, “relfc — Release a pool-file record address” on page 150.

2.64 ronic — Return information about a DASD record or record type

TPF compatibility

Do not use ronic in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int ronic(enum ronic_type type, (void *) file_address_ptr,
          (void **) info_ptr, (void *) ordinal_ptr);
```

type

Specifies what type of processing you require. Use one of the following values:

RONIC_RONINFO

You supply the file address, ALCS provides information about the record in a specified format:

file_address_ptr

Pointer to a location that contains a 4-byte (long int) file address. On return, this is unchanged unless ALCS has detected an error, in which case ALCS sets this field to zero.

info_ptr

Pointer to a pointer to the address of the ronic information area. On return, ALCS loads this area with information relating to the record and file type.

ordinal_ptr

Pointer to an *int*. ALCS sets this, on return, to the record ordinal number. ALCS sets it to an error code if the file address is zero.

RONIC_GETNAME

You supply the file address, ALCS returns information about the record type. Parameters are as follows:

file_address_ptr

Load this with a pointer to a location that contains a 4-byte (long int) file address. On return, this is unchanged unless ALCS has detected an error, in which case ALCS sets this field to zero.

info_ptr

On return, ALCS sets this as a pointer to a pointer to a character string. This string contains the name of the record type. It is not necessarily delimited by a zero byte.

ordinal_ptr

On return, ALCS sets this to a pointer to an *int* field containing the length of the record type name. ALCS sets this to an error code when the file address is zero. (See “Error return”.)

RONIC_GETRON

You supply the record type and ordinal number, ALCS returns the file address. The parameters are as follows:

file_address_ptr
Pointer to the location which is to receive the file address. On return, ALCS sets this to the 4-byte file address, or to zero when an error occurs.

info_ptr
Pointer to a pointer to the record type name. The record type name is set up by the program. Terminate the name with any non-alphabetic character (for example, `'\0'`). On return, ALCS sets *info_ptr* to point to the `ronic` information area that the function provides.

ordinal_ptr
Pointer to a pointer which is set to the record ordinal number as decimal text. Terminate the number with any non-alphanumeric text, (for example, `'\0'`). On return, ALCS stores the ordinal number over the first 4 bytes of the record ordinal number text. When the file address is zero ALCS stores an error code.

Description

Use the `ronic` function in one of three ways:

1. You supply the file address, ALCS provides information about the record in a standard format in the `ronic` information area.
2. You supply the file address, ALCS provides the record type name and the record ordinal number.
3. You supply the record type and ordinal number, ALCS provides the file address.

The *info_ptr* parameter is a pointer to a pointer and is always of type `void**`.

Normal return

The function returns:

RONIC_RONINFO	Record ordinal number
RONIC_GETNAME	Length of the record type name
RONIC_GETRON	Record ordinal number

Error return

Zero. ALCS sets an error code in the *ordinal_ptr* field as follows:

RONIC_RONERRT	Invalid input, unable to determine file address.
RONIC_RONERRG	General file (or general data set) address correct but file not online.
RONIC_RONERRO	Relative record number is too large for this record type.

Loss of control

This function does not cause the entry to lose control.

Example

The following examples show 3 uses of `ronic`. Two of the examples use a common subroutine `show_ron_inf()`:

```
/*====================================================================*/
/* subroutine to display the ronic information area                   */
/*====================================================================*/

void show_ron_inf(ronic_info_area *p)
{
  #define YN(x) ((p->x)?"Yes":"No ")
  printf("record class.......................x'%04X'\n", p->roncls );
  printf("number of records allocated.........%d\n",      p->ronnbr );
  printf("block type code (L1, L2, ...).......x'%04X'\n", p->ronbty );
  printf("logical block length (bytes)........%d\n",      p->ronbln );
  printf("record type/pool interval/gf number.x'%04X'\n", p->rontyp );
  printf("delayed file.......................%s\n", YN(ronvfod ));
  printf("permanently resident...............%s\n", YN(ronvfop ));
  printf("time initiated file................%s\n", YN(ronvfot ));
  printf("force read.........................%s\n", YN(ronvfof ));
  printf("update (read before write) mode.....%s\n", YN(ronvfou ));
  printf("long term pool stamp processing.....%s\n", YN(ronvfol ));
  printf("hyperspace backing.................%s\n", YN(ronvfoh ));
  printf("configuration data set.............%s\n", YN(ronindc ));
  printf("long term pool.....................%s\n", YN(ronindlt));
  printf("short term pool....................%s\n", YN(ronindst));
  printf("pool record........................%s\n", YN(ronindp ));
  printf("general file/dataset record.........%s\n", YN(ronindg ));
  printf("data base record...................%s\n", YN(ronindr ));
}
```

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*  Example of using ronic() to get information about a file address  */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include <tpfeq.h>
#include <tpfapi.h>

 void show_ron_inf(ronic_info_area *);
 int native_file_address=0, ron=0, rc=0;
 ronic_info_area *info_area_ptr;

 native_file_address = 0x000C0840;
 rc = ronic(RONIC_RONINFO,  &native_file_address,
                  (void**)  &info_area_ptr, &ron);

if (rc)
{
  printf("Native file address = %08X\n",native_file_address);
  printf("Record ordinal number = %08X\n",ron);
  printf("Info area is at %08X\n",info_area_ptr);
  show_ron_inf(info_area_ptr);
}
else
{
  printf("Error occured ");

  switch(rc)
  {
    RONIC_RONERRT: printf("Invalid input");
                   break;

    RONIC_RONERRG: printf("General file is offline");
                   break;

    RONIC_RONERRO: printf("Record ordinal number too large");
                   break;

    default:       break;
  }
  printf("\n");
}
```

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*  Example of using ronic() to get the record type and ordinal       */
/*  number for a particular file address                              */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include <tpfeq.h>
#include <tpfapi.h>

 int native_file_address=0, ron=0, rc=0;
 int name_length=0;
 char *name_ptr;

 native_file_address = 0x000C0840;
 rc = ronic(RONIC_GETNAME,  &native_file_address,
                  (void**)  &name_ptr, &name_length);

if (rc)
{
  printf("Native file address = %08X\n",native_file_address);
  printf("Record type name is: %.*s\n", name_length,name_ptr);
  printf("Record ordinal number = %08X\n",ron);
}
else
{
  printf("Error occured ");

  switch(rc)
  {
    RONIC_RONERRT: printf("Invalid input");
                   break;

    RONIC_RONERRG: printf("General file is offline");
                   break;

    RONIC_RONERRO: printf("Record ordinal number too large");
                   break;

    default:       break;
  }
  printf("\n");
}
```

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*  Example of using ronic() to get the file address when supplied    */
/*  with the record type name and the record ordinal number           */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include <tpfeq.h>
#include <tpfapi.h>

 void show_ron_inf(ronic_info_area *);
 int native_file_address=0, ron =0, rc=0;
 ronic_info_area *info_area_ptr;
 char name[8],number[8];

/* set up to get the address on record ordinal number 12 (decimal)    */
/* on the record type "#CANRI"                                        */

 info_area_pointer = (void*) strcpy(name,"#CANRI");
 ron = (int) strcpy(number,"12");

 rc = ronic(RONIC_GETRON,  &native_file_address,
         (void**)  &info_area_ptr , &ron);

if (rc)
{
  printf("Native file address = %08X\n",native_file_address);
  printf("Info area is at %08X\n",info_area_ptr);
  show_ron_inf(info_area_ptr);
}
else
{
  printf("Error occured ");

  switch(rc)
  {
    RONIC_RONERRT: printf("Invalid input");
                   break;

    RONIC_RONERRG: printf("General file is offline");
                   break;

    RONIC_RONERRO: printf("Record ordinal number too large");
                   break;

    default:       break;
  }
  printf("\n");
}
```

Related information

1.20, “<tpfeq.h>” on page 20.
2.71, “sonic — Get symbolic file address information” on page 174.

2.65 routc — Route a message to a communication resource

Format

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$rc0pl.h>

void routc(struct rc0pl *rcpl, enum t_lvl level);
```

rcpl
Pointer to a routing control parameter list (RCPL). This structure is named RC0PL and is defined in header <c$rc0pl.h>.

level
An ECB storage level (D0, D1,...DF), see2.1, “Common parameters” on page 25. The message you want to be transmitted is located on this level. When bit RCPL0EXT is off in the RCPL, the message is in a storage bock attached on the storage level, in standard format. When bit RCPL0EXT is on in the RCPL, the data level contains the address of a heap storage area containing the message, in extended format.

If the message is in a storage block, routc releases it. If the message is in a heap storage area, routc does not free it.

TPF compatibility

routc using heap storage is not supported in TPF. Do not use it in programs that must be compatible with TPF.

Description

Use the routc function to route a message to a terminal or to another (or the same) application. The destination terminal or application can be owned (hosted) by the same ALCS system as the originating application, or by a system in a different processor.

You can also use routc to route a message to:

- An X.25 PVC
- An LU 6.1 link or parallel session
- An APPC connection
- A TCP/IP connection

Messages sent to applications must be in application message (AMSG) format.

Messages sent to terminals can be either in AMSG or OMSG format. You must set the indicator (rcplctl2) in the routing control parameter list (RCPL) to indicate which format you are using.

For details of the message formats and RCPL fields, see 1.9, “ALCS header files” on page 7.

On return, ALCS releases the storage block on the specified level and it is no longer available to the application program.

Attention

If you use routc to send a reply message to the originating terminal, you must not write to the stdout file either explicitly or implicitly (for example, by using the putchar, puts, printf, or vprintf functions in your program).

Normal return
Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Example
This example shows the use of routc to send an unsolicited message.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$rc0pl.h>

struct rc0pl *rp;             /*              pointer to RCPL  */
struct cm1cm *cm;             /*    pointer to output message  */

char msg[80];                /*      buffer for message text  */
int i,cri,orig;

strcpy(msg,"Test message");   /*               message to send  */
cri = 0x020088;               /*            CRI of destination  */

/* build a message block on level D7 */

cm = getcc(D7,GETCC_TYPE,L1); /*                   get a block  */
i  = strlen(msg);             /*    get length of message text  */
msg[i++]  =  _EOM;            /*      add _EOM to message text  */
memcpy(cm->cm1txt,msg,i);     /* copy message and _EOM to block */
cm->cm1cct = i + 5;           /*    set count in message block  */

/* set up an RCPL  */
orig = ecbptr()->ebrout;      /* get my address for origin      */
rp = (struct rc0pl *) ecbptr()->ce1rcpl; /* use the ECB's RCPL  */
memset(rp,0,sizeof(ecbptr()->ce1rcpl));  /* clear the RCPL      */
memcpy(&rp->rcpldes,((char *)(&cri))+1,3);/* destination in RCPL*/
rp->rcplctl0 |=  RCPL0MTY;       /* set as unsolicited message  */
memcpy(&rp->rcplorg,((char*)(&orig)+1,3); /* origin in RCPL     */

routc(rp,D7);                 /*           send the message     */
```

Related information
1.8, “Routing of messages” on page 6.
1.16.2, “RCPL required for an output message” on page 16.

2.66 serrc_op — Request an error dump

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void serrc_op(enum t_serrc status, int number, const char *msg,
              void *storage_list[]);
```

status
Specifies whether you want control to return to the entry following the dump. Specify SERRC_EXIT to force the entry to exit, or SERRC_RETURN to return control to the entry.

number
The number of the system error dump. Use a unique number in the range 0x001000 through 0xFFFFFF (0x000000 through 0x000FFF are reserved for ALCS). Some other ranges of numbers might be reserved in your installation. For example, numbers starting 0xDB.... are reserved for TPFDF. (If you specify a number greater than 0xFFFFFF, ALCS uses only the last 24 bits of the number.)

msg
Pointer to a message text string that ALCS appends to the system error message. The string must not exceed 255 characters. (If it does, ALCS uses the first 255 characters and truncates the rest.) When you want no message to be printed, code as NULL.

storage_list
This parameter is provided for compatibility with TPF, code as NULL. ALCS ignores it. If you need to dump an area of storage, use `snapc` or `serrc_op_ext`.

Description

You can use the `serrc_op` function to cause ALCS to issue a system error dump. The dump has an identifying number and optionally includes a message. You can choose to exit the entry or to return to it after the dump.

If you need to dump an area of storage, use `snapc` or `serrc_op_ext` instead of `serrc_op`.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

Example

This example generates a system error dump, number 012345, when `waitc` detects an error. ALCS appends the text, "Error occurred" to the system error dump and returns control to the program after the dump.

```
#include <tpfeq.h>
#include <tpfapi.h>

  ⋮
if (waitc())
   serrc_op(SERRC_RETURN,0x12345,"Error occurred",NULL);
  ⋮
```

Related information

C/C++ Run-Time Library Reference – functions `abort`, `exit`, `perror`.

2.67, "`serrc_op_ext` — Request an error dump, with extended options" on page 163.

2.70, "`snapc` – Request an error dump" on page 171.

2.67 serrc_op_ext — Request an error dump, with extended options

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void serrc_op_ext(enum t_serrc status, int number, const char *msg,
                  char prefix, struct serrc_list **list);
```

status
Specifies whether you want control to return to the entry following the dump. Specify SERRC_EXIT to force the entry to exit, or SERRC_RETURN to return control to the entry.

number
The number of the system error dump. Use a unique number in the range 0x001000 through 0xFFFFFF (0x000000 through 0x000FFF are reserved for ALCS). Some other ranges of numbers might be reserved in your installation. For example, numbers starting 0xDB.... are reserved for TPFDF. (If you specify a number greater than 0xFFFFFF, ALCS uses only the last 24 bits of the number.)

msg
Pointer to a message text string that ALCS appends to the system error message. The string must not exceed 255 characters. (If it does, ALCS uses the first 255 characters and truncates the rest.) When you want no message to be printed, code as NULL.

prefix
This parameter is provided for compatibility with TPF, code as 'U' or any other 1-character code. ALCS ignores it.

TPF compatibility

In TPF you can specify a 1-character prefix for the dump number but IBM reserves prefixes 'I' and 'W' through 'Z', for system use.

list Pointer to an array of pointers that point to a serrc_list structure (#defined in <tpfapi.h>), which indicate the areas of storage you want to be displayed in the dump.

The serrc_list structure has four fields:

serrc_len
The length of the area to be dumped. Code as zero to indicate no more areas are to be dumped.

serrc_name
Name you want to use to identify the data area. *serrc_name* must be 8-characters long, left justified, and padded with blanks.

serrc_tag
The location of the area to be dumped.

serrc_indir
Indicates whether the address in *serrc_tag* is an indirection. To indicate an indirection, code the value SERRC_INDIR. If *serrc_tag* is not an indirection, code the value SERRC_NOINDIR.

ALCS allows a maximum of 50 elements in a *serrc_list* array.

If you do not want to display a storage list, code *list* as NULL.

Description

Use the `serrc_op_ext` function to cause ALCS to issue a system error dump. The dump has an identifying number and optionally includes a message and various areas of storage. You can choose to exit the entry or to return to it after the dump.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

Example

This example produces a system serror dump with dump number AAAA11. After the dump, ALCS exits the entry. The message displayed on the dump is:

```
Test of serrc_op_ext with user list
```

Two additional data areas to be dumped are specified with a *list* parameter:

- The ECB work area; this is addressed directly.
- The block on level 0; this is addressed indirectly using data level 0.

```
/*-------------------------------------------------*/
/* initialize the array of pointers                */
/*-------------------------------------------------*/
serrclist[0] = &serrcitem[0];
serrclist[1] = &serrcitem[1];
serrclist[2] = &serrcitem[2];

/*-------------------------------------------------*/
/* dump the ecb work area                          */
/*-------------------------------------------------*/
serrcitem[0].serrc_len     = sizeof(ecbptr()->ce1wka);
serrcitem[0].serrc_name    = "ECB WORK";
serrcitem[0].serrc_tag     = &ecbptr()->ce1wka;
serrcitem[0].serrc_indir   = SERRC_NOINDIR;

/*-------------------------------------------------*/
/* dump the contents on the block on level 0       */
/*-------------------------------------------------*/
serrcitem[1].serrc_len     = levtest(D0);
serrcitem[1].serrc_name    = "LEVEL D0";
serrcitem[1].serrc_tag     = &ecbptr()->ce1cr0;
serrcitem[1].serrc_indir   = SERRC_INDIR;

/*-------------------------------------------------*/
/* terminate list with a zero length in last item */
/*-------------------------------------------------*/
serrcitem[2].serrc_len     = 0;

/*-------------------------------------------------*/
/* call serrc_op_ext to dump with exit             */
/*-------------------------------------------------*/
serrc_op_ext(SERRC_EXIT, 0x0AAAA11,
         "Test of serrc_op_ext with user list",
          'U',
          serrclist );
```

Related information

C/C++ Run-Time Library Reference – functions `abort`, `exit`, `perror`.

2.70, “`snapc` – Request an error dump” on page 171.
2.66, “`serrc_op` — Request an error dump” on page 161.

2.68 serrc_op_slt — Request an error dump, with storage list

TPF compatibility

ALCS provides this function only for compatibility with TPF. IBM recommends you to use snapc or serrc_op_ext instead of serrc_op_slt in ALCS programs.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void serrc_op_slt(enum t_serrc status, int number, const char *msg,
                  void *slist[], char prefix);
```

status
Specifies whether you want control to return to the entry following the dump. Specify SERRC_EXIT to force the entry to exit, or SERRC_RETURN to return control to the entry.

number
The number of the system error dump. Use a unique number in the range 0x001000 through 0xFFFFFF (0x000000 through 0x000FFF are reserved for ALCS). Some other ranges of numbers might be reserved in your installation. For example, numbers starting 0xDB.... are reserved for TPFDF. (If you specify a number greater than 0xFFFFFF, ALCS uses only the last 24 bits of the number.)

msg
Pointer to a message text string that ALCS appends to the system error message. The string must not exceed 255 characters. (If it does, ALCS uses the first 255 characters and truncates the rest.) When you want no message to be printed, code as NULL.

slist
This parameter is provided for compatibility with TPF, code as NULL. ALCS ignores it. If you need to dump an area of storage, use snapc or serrc_op_ext.

prefix
This parameter is provided for compatibility with TPF, code as 'U' or any other 1-character code. ALCS ignores it.

TPF compatibility

In TPF you can specify a 1-character prefix for the dump number but IBM reserves prefixes 'I' and 'W' through 'Z', for system use.

Description

You can use the serrc_op_slt function to cause ALCS to issue a system error dump. The dump has an identifying number and optionally includes a message. You can choose to exit the entry or to return to it after the dump.

Normal return
Void.

Error return
Not applicable.

Loss of control
This function causes the entry to lose control.

Example
No example provided. IBM recommends you not to use this function in ALCS.

Related information
C/C++ Run-Time Library Reference – functions `abort`, `exit`, `perror`.
2.70, “`snapc` – Request an error dump” on page 171.
2.67, “`serrc_op_ext` — Request an error dump, with extended options” on page 163.

2.69 slimc — Set or remove processing limits for the entry

TPF compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void slimc(enum slimc_type type, int value
           [,enum slimc_units units]);
```

type

Specifies which processing values you want to set. (You set the actual values in the *value* parameter.)

SLIMC_GFS
: The maximum number of short-term pool file dispenses and the maximum number of long-term pool file dispenses (getfcs). Specify a value of 1 through 65 535 (0xFFFF) or SLIMC_NONE for the maximum system limit.

SLIMC_STPOOL
: The maximum number of short-term pool file dispenses (getfcs). Specify a value of 1 through 65 535 (0xFFFF) or SLIMC_NONE for the maximum system limit.

SLIMC_LTPOOL
: The maximum number of long-term pool file dispenses (getfcs). Specify a value of 1 through 65 535 (0xFFFF) or SLIMC_NONE for the maximum system limit.

SLIMC_SERR
: The maximum number of system error dumps and nodumps. Specify a value of 1 through 65 535 (0xFFFF) or SLIMC_NONE for the maximum system limit.

SLIMC_ECBLIFE
: The maximum entry life. (The units you are using for life are defined in the *units* parameter.) Specify a value of 1 through 65 535 (0xFFFF) seconds or 1 through 1092 minutes, or 1 through 18 hours, or SLIMC_NONE for the maximum system limit.

SLIMC_ECBREAD
: Entry read threshold. ALCS calls the USRWAIT installation-wide monitor exit during an implied wait operation each time the number of reads for this entry exceeds one of the thresholds. Specify from 1K to 65 535K reads (where K is 1024), otherwise specify SLIMC_NONE for the maximum threshold.

SLIMC_ECBHOLD
: Maximum number of records that can held at the same time. Specify a value of 1 through 65 535 (0xFFFF), otherwise specify SLIMC_NONE for the maximum system limit.

SLIMC_STORAGE
: The maximum amount of storage that an entry can acquire for each storage unit type. Specify a value of 1 through 67 107 840 bytes or 1 through 65 535 kilobytes, or 1 through 64 megabytes, otherwise SLIMC_NONE for the maximum system limit.

  IBM recommends that you do not use SLIMC_STORAGE; use SLIMC_STORAGE1, SLIMC_STORAGE2, and SLIMC_STORAGE3 instead.

  Note that SLIMC_STORAGE is equivalent to SLIMC_STORAGE1 plus SLIMC_STORAGE2 plus SLIMC_STORAGE3.

SLIMC_STORAGE1
: Maximum entry storage for ECB, attached and detached storage blocks, automatic storage blocks, and DECBs. Specify a value of 1 through 67 107 840 bytes or 1 through 65 535 kilobytes, or 1 through 64 megabytes, otherwise SLIMC_NONE for the maximum system limit.

SLIMC_STORAGE2
: Maximum entry storage for high level language (HLL) stack storage. Specify a value of 1 through 67 107 840 bytes or 1 through 65 535 kilobytes, or 1 through 64 megabytes, otherwise SLIMC_NONE for the maximum system limit.

SLIMC_STORAGE3
: Maximum entry storage for program heap storage. Specify a value of 1 through 67 107 840 bytes or 1 through 65 535 kilobytes, or 1 through 64 megabytes, otherwise SLIMC_NONE for the maximum system limit.

value
: Specifies the actual new value you want for for the selected limit. Use the value SLIMC_NONE to remove an existing limit.

units
: This parameter is optional.

  For SLIMC_ECBLIFE use one of:

SLIMC_HOURS	value in hours (default)
SLIMC_MINUTES	value in minutes
SLIMC_SECONDS	value in seconds.

  For SLIMC_STORAGE, SLIMC_STORAGE1, SLIMC_STORAGE2, and SLIMC_STORAGE3 use one of:

SLIMC_MEGABYTES	value in megabytes (default)
SLIMC_KILOBYTES	value in kilobytes
SLIMC_BYTES	value in bytes.

  For SLIMC_GFS, SLIMC_ERR, and SLIMC_NONE, this parameter is ignored.

Description

Use `slimc` to set one or more upper limits of the processing resources for an entry. You can also use `slimc` to remove the limits.

Normal return
None.

Error return
None.

Loss of control
This function does not cause the entry to lose control.

Example
This example removes the limit on the number of pool file dispenses and sets the maximum time for an entry to 8 minutes.

```
#include <tpfeq.h>
#include <tpfapi.h>

slimc(SLIMC_GFS,SLIMC_NONE);        /* no limit on pool file dispense*/
slimc(SLIMC_ECBLIFE,8,SLIMC_MINUTES); /* max entry life is 8 minutes */
```

Related information
2.56, “longc — Set maximum entry life” on page 138.

2.70 snapc — Request an error dump

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void snapc(int status, int number, const char *msg,
     struct snapc_list **listc, char prefix,
     int regs, int ecb, char *program);
```

status
Specifies whether you want control to return to the entry after the dump. Use SNAPC_EXIT to force the entry to exit, or SNAPC_RETURN to return control to the entry.

number
The number of the system error dump. Use a unique number in the range 0x001000 through 0xFFFFFF (0x000000 through 0x000FFF are reserved for ALCS). Some other ranges of numbers might be reserved in your installation. For example, numbers starting 0xDB.... are reserved for TPFDF. (If you specify a number greater than 0xFFFFFF, ALCS uses only the last 24 bits of the number.)

msg
Pointer to a message text string that ALCS appends to the system error message. The string must not exceed 255 characters. (If it does, ALCS uses the first 255 characters and truncates the rest.) When you want no message to be printed, code as NULL.

listc
Pointer to an array of pointers that point to a `snapc_list` structure (`#defined` in `<tpfapi.h>`, which indicate the areas of storage you want to be displayed in the dump.

The `snapc_list` structure has four fields:

snapc_len
The length of the area to be dumped. Code as zero to indicate no more areas are to be dumped.

snapc_name
Name you want to use to identify the data area. *snapc_name* must be 8-characters long, left justified, and padded with blanks.

snapc_tag
The location of the area to be dumped.

snapc_indir
Indicates whether the address in *snapc_tag* is an indirection. To indicate an indirection, code the value SNAPC_INDIR. If *snapc_tag* is not an indirection, code the value SNAPC_NOINDIR.

ALCS allows a maximum of 50 elements in a *snapc_list* array.

If you do not want to display a storage list, code *listc* as NULL.

prefix
This parameter is provided for compatibility with TPF, code as 'U' or any other 1-character code. ALCS ignores it.

TPF compatibility

In TPF you can specify a 1-character prefix for the dump number but IBM reserves prefixes 'I' and 'W' through 'Z', for system use.

regs
This parameter is provided for compatibility with TPF. ALCS ignores it. Code it as SNAPC_REGS or SNAPC_NOREGS.

ecb
This parameter is provided for compatibility with TPF. ALCS ignores it. Code as SNAPC_NOECB or SNAPC_ECB.

program
Pointer to a 4-character program name which ALCS displays in the dump. Code as NULL when you want the current program name to appear in the dump.

TPF compatibility

TPF allows a program name up to 16 characters long. The name is terminated by a binary zero character ('\0').

Description
Use the snapc function to cause ALCS to issue a system error dump. This dump has an identifying number and optionally includes a message and various areas of storage. You can choose to exit the entry or to return to it after the dump.

Normal return
Void.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Example

This example generates a system error dump with dump number 012345. ALCS returns control to the program after the dump. The program name is TEST. The *snapc_list* is `snapstuff` and the message is “Program failed”.

```
#include <tpfeq.h>
#include <tpfapi.h>

test()
{
 struct snapc_list *snapstuff[3],list[3]

 snapstuff[0]=&list[0];
 snapstuff[1]=&list[1];
 snapstuff[2]=&list[2];

 snapstuff[0]->snapc_len = 4;
 snapstuff[0]->snapc_name = "MYSTUFF ";
 snapstuff[0]->snapc_tag = &ecbptr()->ebw000;
 snapstuff[0]->snapc_indir = SNAPC_INDIR;

 snapstuff[1]->snapc_len = 1000;
 snapstuff[1]->snapc_name = "DATA10  ";
 snapstuff[1]->snapc_tag = &ecbptr()->ce1cr0;
 snapstuff[1]->snapc_indir = SNAPC_INDIR;

 snapstuff[2]->snapc_len = 0;

 snapc(SNAPC_RETURN,0x12345,"Program failed",snapstuff,
       'A',SNAPC_REGS, SNAPC_ECB, "TEST");

 exit(0):
}
```

Related information

C/C++ Run-Time Library Reference – functions `abort`, `exit`, `perror`.
2.67, “`serrc_op_ext` — Request an error dump, with extended options” on page 163.

2.71 sonic — Get symbolic file address information

TPF compatibility

ALCS provides this function for compatibility with TPF. IBM recommends you not to use this function in ALCS, use ronic instead.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

unsigned long sonic(enum t_lvl level)
```

or

```
#include <tpfeq.h>
#include <tpfapi.h>

unsigned long sonic(TPF_FA8 *fa8)
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the ECB data level containing the file address.

fa8
A pointer to an 8-byte file address.

Description

You can use the sonic function to return information about a file address.

Note: Applications that call this function using an 8-byte file address instead of an ECB data level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

The return value of sonic contains non-exclusive bit settings which you can test using the following defined values:

SONIC_TYPE_INDICATOR
If set, the record is a pool-file record; if unset, the record is a fixed-file record.

SONIC_POOL_LONGEVITY
If set, the record is a short-term pool-file record; if unset, the record is a long-term pool-file record (SONIC_TYPE_INDICATOR must be set).

SONIC_FADDR_INVALID
If set, the file address is invalid; if unset, the file address is valid.

SONIC_REC_SIZE
If set, the record size is at least 1055 bytes; if unset, the record size is less than 1055 bytes.

For compatibility with TPF, you can also test for the following bit settings: SONIC_ADDRESS_FORMAT, SONIC_FARF4_ADDRESS, SONIC_FARF5_ADDRESS, SONIC_RECORD_UNIQUE, SONIC_ALT_REC_SIZE, and SONIC_DUPLICATED. In ALCS, these are always unset.

Error return

`sonic` returns the value SONIC_ERROR.

Loss of control

This function does not cause the entry to lose control.

Examples

This example returns information about the file address in ECB data level 4 (D4).

```
#include <tpfeq.h>
#include <tpfapi.h>

unsigned long ret_code;
  ⋮
ret_code = sonic(D4);

if (ret_code == SONIC_ERROR)
{
  /* error */
}

if (ret_code & SONIC_FADDR_INVALID)
{
  /* invalid file address */
}
else
{
  if (ret_code & SONIC_TYPE_INDICATOR)
  {
    /* pool-file record */
    if (ret_code & SONIC_POOL_LONGEVITY)
    {
      /* short-term pool-file record */
    }
    else
    {
      /* long-term pool-file record */
    }
  }
  else
  {
    /* fixed-file record */
  }
  if (sonic_return & SONIC_REC_SIZE)
  {
    /* record size >= 1055 bytes */
  }
  else
  {
    /* record size < 1055 bytes */
  }
}
```

This example returns information about an 8-byte file address.

```
#include <tpfeq.h>
#include <tpfapi.h>

unsigned long ret_code;
TPF_FA8 fa8;
  ⋮
ret_code = sonic(fa8);
```

Related information

2.64, “ronic — Return information about a DASD record or record type” on page 153.

2.72 tape_close — Close a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tape_close(char *name);
```

name
Pointer to the 3-character name of the sequential file to be closed.

Description

This function closes a general sequential file. The sequential file must have been opened using tape_open and must **not** be assigned to an entry.

After it is closed, the sequential file is unavailable to the application program.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

The following example closes a sequential file with the name VPH.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tape_close("VPH");
```

Related information

2.73, “tape_open — Open a general sequential file” on page 178.
2.74, “tape_read — Read a record from a general sequential file” on page 179.
2.75, “tape_write — Write a record to a general sequential file” on page 181.

2.73 tape_open — Open a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tape_open(char *name, int io);
```

name
Pointer to the 3-character name of the sequential file to be opened.

io Specifies whether you are opening the file for input or output. Use the value INPUT to open the file for reading records and and OUTPUT to open the file for write operations.

Description

This function makes a general sequential file available to the application program for reading or writing (not both).

The file must not be open at the time of the call.

Normal return

Void. The specified sequential file is positioned at the first record, and is in the reserved state, meaning it is available for use by any entry.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example opens a sequential file VPH for output.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tape_open("VPH", OUTPUT);
```

Related information

2.72, “tape_close — Close a general sequential file” on page 177.
2.74, “tape_read — Read a record from a general sequential file” on page 179.
2.75, “tape_write — Write a record to a general sequential file” on page 181.

2.74 tape_read — Read a record from a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void *tape_read(char *name, enum t_lvl level,
                enum t_blktype size);
```

name
Pointer to the 3-character name of the sequential file to be read from.

level
A storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the storage level to which ALCS attaches the storage block containing the sequential file record. (This storage level must not have an attached storage block when the function is called.

size
This is a block size code: L0, L1, ...L8, as defined in your installation.

Description

This function reads a record from a general sequential file. It reads the record into a storage block of the appropriate size and attaches this block to the ECB at the specified *level*. The sequential file must have been opened using `tape_open`, and must **not** be currently assigned to the issuing entry.

The function suspends the entry until the read is complete. You can test for errors by checking the return value of the function being NULL.

On completion of the function, ALCS leaves the sequential file in the reserved state, making it available for use by any entry.

Normal return

Pointer to the storage block containing the sequential file record image.

Error return

NULL if this or any outstanding I/O operation was unsuccessful.

Detailed error information is provided in the error byte for data level *n* (`ecbptr()->ce1sud[n]`). See *ALCS Application Programming Guide*.

Loss of control

This function can cause the entry to lose control.

The return value of the function tells the application whether the read was successful. See “Error return”.

Example

This example opens, reads a record from, and closes sequential file VPH.

```
#include <tpfeq.h>
#include <tpftape.h>

struct vp0vp *vp;
  ⋮
tape_open("VPH",INPUT);

if ((vp = (struct vp0vp *) tape_read("VPH",D6,L2)) == NULL)
  exit(0x56789);              /*   Dump & exit if read failed  */
tape_close("VPH");            /*   Close it.                   */
```

Related information

2.72, “tape_close — Close a general sequential file” on page 177.
2.73, “tape_open — Open a general sequential file” on page 178.
2.75, “tape_write — Write a record to a general sequential file” on page 181.

2.75 tape_write — Write a record to a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tape_write(char *name, enum t_lvl level);
```

name
Pointer to the 3-character name of the sequential file you want to write to.

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the storage level on which the record image is attached. ALCS writes this record to the sequential file.

Description

This function writes a record to a general sequential file. ALCS takes the data from a storage block attached at the ECB at the level specified in the *level* parameter. The sequential file must have been opened using tape_open, and must **not** be currently assigned to the issuing entry.

On completion of tape_write, ALCS leaves the sequential file in the reserved state, making it available for use by any entry. The status of the I/O operation is not returned to the application program.

ALCS releases the storage block on the storage level specified in the *level* parameter. The block is no longer available to any application program.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the write was successful.

Example

The following example opens and writes a copy of the record on level 1 (D1) to the sequential file VPH.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tape_open("VPH", OUTPUT);
tape_write("VPH", D1);     /*  Write block on D1 to tape     */
tape_close("VPH");
```

Related information

2.72, “tape_close — Close a general sequential file” on page 177.
2.73, “tape_open — Open a general sequential file” on page 178.
2.74, “tape_read — Read a record from a general sequential file” on page 179.

2.76 tasnc — Assign a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tasnc(char *name);
```

name
Pointer to a 3-character name of the sequential file name.

Description

Use the tasnc function to assign a general sequential file to the entry. The entry has exclusive use of the file until it closes it by calling tclsc, or unassigns it by calling trsvc.

The general sequential file must be open but can be assigned to another entry. In this case, ALCS suspends the entry until the other entry reserves the file.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

Example

This example assigns sequential file VPH to the current entry.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tasnc("VPH");
```

Related information

2.77, “tclsc — Close a general sequential file” on page 184.
2.80, “topnc — Open a general sequential file” on page 191.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.93, “trsvc — Reserve a general sequential file” on page 215.
2.94, “twrtc — Write a record to a general sequential file” on page 216.

2.77 tclsc — Close a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tclsc(char *name);
```

name
Pointer to the 3-character name of the general sequential file to be closed.

Description

Use the tclsc function to close a general sequential file.

The general sequential file must be open, and assigned to the issuing entry.

After tclsc, the general sequential file is unavailable to the application program.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the close was successful.

Example

The following example closes a sequential file VPH.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tclsc("VPH");
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.80, “topnc — Open a general sequential file” on page 191.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.93, “trsvc — Reserve a general sequential file” on page 215.
2.94, “twrtc — Write a record to a general sequential file” on page 216.

2.78 tdspc — Get information about any type of sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

struct tpstat *tdspc(char *name, char type, enum t_lvl level);
```

name
Pointer to the 3-character name of the sequential file whose status you require. The sequential file need not be assigned to the issuing entry.

type
Specifies whether you want the status of the active or standby sequential file. Use ACTIVE to specify the active sequential file, or STANDBY to specify the standby sequential file.

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the data level on which ALCS puts the status information.

Description

Use the tdspc function to extract information from the sequential file configuration table.

tdspc puts the information in a tpstat structure defined in <tpftape.h>. (See 1.23, “<tpftape.h>” on page 23.)

Normal return

Pointer to structure tpstat (defined in <tpftape.h>) which contains the sequential file status.

Error return

NULL.

Loss of control

This function does not cause the entry to lose control.

Example

This example calls tdspc to examine the status of the active sequential file VPH.

```
#include <tpfeq.h>
#include <tpftape.h>

struct tpstat *status;
  ⋮
if ((status = tdspc("VPH", ACTIVE, D0)) == NULL)
  {
   errno = 0x1234;
   perror("VPH TAPE NOT MOUNTED");
   abort();
  }
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.77, “tclsc — Close a general sequential file” on page 184.
2.80, “topnc — Open a general sequential file” on page 191.
1.23, “<tpftape.h>” on page 23.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.93, “trsvc — Reserve a general sequential file” on page 215.
2.94, “twrtc — Write a record to a general sequential file” on page 216.

2.79 timec — Get time and date

TPF compatibility

Do not use this function in programs that must be compatible with TPF.

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

void * timec(enum timec_type type, enum timec_format format,
             enum timec_zone1 zone1, enum timec_zone2 zone2,
             (void *) result_ptr);
```

type

The type of time or date information you require. Use one of the following:

TIMEC_TIMEDATE
: Time and date, separated by one blank, in character form.

TIMEC_TIME
: Time in character form.

TIMEC_DATE
: Date in character form.

TIMEC_TIMEDIFF
: Difference between GMT and local time.

TIMEC_SECONDS
: Perpetual seconds clock value. The number of seconds since the last ALCS restart.

TIMEC_MINUTES
: Perpetual minutes clock value. The number of minutes since 00:00 hours on 3rd January 1966 local time.

format

The format required for the time or date when TIMEC_TIMEDATE, TIMEC_TIME, or TIMEC_DATE are specified.

TIMEC_JULIAN
: 24 hour clock “YYDDD”

TIMEC_DMY
: Day-month-year “DD.MM.YY”

TIMEC_MDY
: Month-day-year “MM.DD.YY”

TIMEC_YMD
: Year-month-day “YY.MM.DD”

TIMEC_INSTALL
: Installation default date, or time or both time and date.

  The target area pointer (*result_ptr*) must point to a 2-byte length field immediately followed by the field where the result will be placed. The

length field is updated to contain the true length of the result returned. If the length field is too small the default format string is truncated.

TIMEC_CUSTOM
Customized date and time format. The target area pointer (*result_ptr*) must point to a 2-byte length field immediately followed by the format string itself. The length field is updated to contain the true length of the result returned. The format string can contain one or more of the following tokens separated (optionally) by delimiter characters.

DD Decimal day number, 2 characters.

MM Decimal month number, 2 characters.

LLL... Month name. These are defined at system generation (see the description of the SCTGEN macro in *ALCS Installation and Customization*). The number of 'L' characters must be at least three and should be at least equal to the number of characters in the longest month name. Any additional 'L' characters will be truncated. If you use too few 'L' characters then the month name will be truncated.

SSS Abbreviated month name. These are defined at system generation (see the description of the SCTGEN macro in *ALCS Installation and Customization*). The number of 'S' characters should be at least three and should be at least equal to the number of characters in the longest abbreviated month name. Any additional 'S' characters will be truncated. If you use too few 'S' characters then the abbreviated month name will be truncated.

YY Decimal year, 2 digits.

YYYY Decimal year, 4 characters.

JJJ Julian day, 3 characters.

hh Hour, 2 digits.

mm Minutes, 2 digits.

ss Seconds, 2 digits.

am Time of day before or after 12:00h (expressed as am or pm), 2 characters.

zone1
With TIMEC_TIMEDATE, TIMEC_TIME, TIMEC_DATE, and TIMEC_TIMEDIFF specifies whether local time and date, or Greenwich Mean Time is to be displayed. Ignored for other types.

TIMEC_LOCAL
Display local time

TIMEC_GMT
Display GMT

In addition you may add the following to achieve 24 or 12 hour displays (the default is 24 hours).

+TIMEC_24_HOUR
24 hour time display

+TIMEC_12_HOUR
12 hour time display

zone2
With TIMEC_TIMEDATE, TIMEC_TIME, and TIMEC_DATE specifies whether ALCS, or the host operating system, time and date is to be displayed. Ignored for other types.

TIMEC_ALCS
Display ALCS time

TIMEC_HOST
Display host operating system (MVS) time

result_ptr
Is a pointer to the area where the result is to be stored. For TIMEC_TIMEDATE, TIMEC_TIME, TIMEC_DATE, TIMEC_INSTALL and TIMEC_CUSTOM a character string is stored. For TIMEC_TIMEDIFF, TIMEC_SECONDS and TIMEC_MINUTES a `long int` is stored.

Description
Use the `timec` function to get time or date information, or both, from ALCS or MVS.

Character strings returned are not delimited by a zero (`'\0'`) character, except for TIMEC_INSTALL and TIMEC_CUSTOM when the output character string is shorter than the input buffer length.

Normal return
Pointer to the result is returned.

Error return
None.

Loss of control
This function does not cause the entry to lose control.

Example
This example displays the local ALCS time and date, the local MVS time and date, and the total time that ALCS has been active. It also includes an example of a customized date format.

```
#include <tpfeq.h>
#include <tpfapi.h>

#define TIME_SIZE 20
#define BUF_SIZE 50
#define BUF_LEN  2

char bufferffBUF_SIZE+BUF_LEN“;
char customffBUF_SIZE“;
char time_buf-TIME_SIZE-;
int  time_int;

memset(time_buf,0,TIME_SIZE);     /* clear buffer */
timec(TIMEC_TIMEDATE,
      TIMEC_YMD,
      TIMEC_LOCAL,
      TIMEC_ALCS,
      time_buf);

printf("Local ALCS time/date: %s\n",time_buf);

memset(time_buf,0,TIME_SIZE);     /* clear buffer */
timec(TIMEC_TIMEDATE,
      TIMEC_YMD,
      TIMEC_LOCAL,
      TIMEC_HOST,
      time_buf);
printf("Local MVS  time/date: %s\n",time_buf);

timec(TIMEC_SECONDS,
      TIMEC_YMD,
      TIMEC_LOCAL,
      TIMEC_ALCS,
      &time_int);

printf("ALCS has been up for %d hrs %d mins %d secs\n",
       time_int/3600,
       (time_int%3600)/60,
       time_int%60 );
⋮


memset(buffer,0,BUF_SIZE+BUF_LEN);

strcpy(custom,"Custom date : DD LLLLLLLLL YYYY");
strcpy(&bufferff2“,custom);
bufferff1“=(char)strlen(custom);

timec(TIMEC_TIME,       /* ignored since CUSTOM request */
      TIMEC_CUSTOM,
      TIMEC_GMT,
      TIMEC_HOST,
      buffer);

printf("%s\n",&bufferff2“);
```

Related information

C/C++ Run-Time Library Reference – `time` function.

2.91, “`tpf_STCK` — Store clock” on page 211.

2.80 topnc — Open a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void topnc(char *name, int io, int bufmode);
```

name
Pointer to the 3-character name of the general sequential file to be opened.

io A value specifying whether the sequential file is to be read (input) or written (output). Use INPUT to denote an input sequential file or OUTPUT to denote an output sequential file.

bufmode
This parameter is used in TPF for 3480 buffering. ALCS ignores it. Use the value NOBUFF to denote no buffering (write immediate mode), or BUFFERED to denote buffered write mode (preferred).

Description

Use the topnc function to open the specified general sequential file and assigns it to the entry.

The general sequential file *name* must not be open at the time of the function call.

Normal return

Void. The specified sequential file is positioned at the first record, and the sequential file has been assigned to the issuing entry.

Error return

Not applicable.

Loss of control

This function causes the entry to lose control.

Example

This example opens the sequential file VPH for input in sequential file write immediate mode.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
topnc("VPH",INPUT,BUFFERED);
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.77, “tclsc — Close a general sequential file” on page 184.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.93, “trsvc — Reserve a general sequential file” on page 215.
2.94, “twrtc — Write a record to a general sequential file” on page 216.
ALCS Installation and Customization (SCTGEN macro and USRSEQ1 installation-wide monitor exit).

2.81 tourc — Write a storage block to a real-time sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tourc(char *name, enum t_lvl level);
```

name
Pointer to the 3-character name of the sequential file to be written to.

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. tourc writes the contents of the storage block on this level as a record to the sequential file.

Description

Use the tourc function to write the storage block attached at the specified storage level to a real-time sequential file. tourc release the storage block after the write, so it is no longer available to the entry.

The status of the operation is not returned to the application program.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the write was successful.

Example

This example writes the storage block on level 9 (D9) to the RTA sequential file. On return, the block is no longer available to the application program.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tourc("RTA",D9);
```

Related information

2.82, “toutc — Write a record to a real-time sequential file” on page 193.

2.82 toutc — Write a record to a real-time sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void toutc(char *name, enum t_lvl level, int bufmode);
```

name
Pointer to the 3-character name of the real-time sequential file to be written to.

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. The data level contains the address and length of the record to be written to the real-time sequential file.

bufmode
This parameter is used in TPF for 3480 buffering. ALCS ignores it. Use the value NOBUFF to denote no buffering (write immediate mode), or BUFFERED to denote buffered write mode (preferred).

Description

Use the `toutc` function to write the record in the specified data level to be a real-time sequential file. The first 4 bytes of the data level specified must contain the storage address of the record to be written. The last 2 bytes of the data level (`ebcfh`*n* and `ebcfr`*n*) must contain the length in bytes of the record (see “Example”).

The `toutc` function does not release the storage. It is available to the entry when ALCS returns control to the entry after a subsequent `waitc`.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a `waitc` call, or with a call to a function that has an implied `waitc`. The return value of the `waitc` lets you check whether the preceding I/O operations, including this read, were successful.

Example

This example writes the storage block on level 9 (D9) to the sequential file RTA. If an I/O error occurs, it produces a dump, number 001234.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
ecbptr()->ce1fa9 = (long int) (ecbptr()->ce1cr9);  /* get address in FARW   */
(*((short int *)&(ecbptr()->ebcfh9))) = 128;       /* length of record      */
toutc("RTA",D9,BUFFERED);
if (waitc())
  {
    errno = 0x1234;
    perror("ERROR WRITING RTA TAPE");
    abort();
  }
```

Related information

2.81, “tourc — Write a storage block to a real-time sequential file” on page 192.

2.83 tpf_decb_create — Create a data event control block

Format

```
#include <tpfeq.h>
#include <c$decb.h>

TPF_DECB *tpf_decb_create(char *name, DECBC_RC *rc);
```

name
A pointer to a 16-byte user-specified DECB name. The *name* parameter is optional. If NULL is coded, a name is not assigned to the DECB.

rc A pointer to the return code. The *rc* parameter is optional. If NULL is coded the return code will not be set.

Description

Use the tpf_decb_create function to allocate storage for a single data event control block (DECB), initialize it and attach it to the ECB.

The DECB is an alternative to an ECB level, which is used to identify a DASD record that is to be read or written and to attach a storage block. The DECB fields specify the same data level and storage level information without requiring the use of an ECB level. All the same requirements and conditions that apply to the data level and storage level in the ECB also pertain to the same fields in the DECB.

The data level in the DECB provides storage for an 8-byte file address in 4x4 format.

The structure of the DECB (defined type TPF_DECB) is defined in <c$decb.h> and described in *ALCS Application Programming Guide*.

The type DECBC_RC is defined in <c$decb.h>.

The functions that support the use of a DECB (for example, file_record_ext and find_record_ext) will only accept a DECB address as a valid reference to a DECB. If an application does not maintain the address of a particular DECB and instead maintains the name of the DECB, the caller must first issue the tpf_decb_locate function to obtain the address of the desired DECB. The resulting DECB address may then be passed on to the subsequent function call.

DECB names beginning with the letters I, i, TPF, TPF_, tpf, and tpf_ are reserved for future use by IBM.

Note: Applications that use DECBs must be compiled with the C++ compiler.

Normal return

A pointer to a DECB (defined type TPF_DECB) and the return code *rc* (defined type DECBC_RC) is set to the defined value DECBC_OK.

Error return

A NULL pointer and the return code *rc* (defined type DECBC_RC) contains the defined value DECBC_DUPNAME if the *name* parameter specified contains a DECB name that already exists for this ECB.

Loss of control

This function does not cause the entry to lose control.

Example

This example creates a DECB with the name "APPLWXY".

```
#include <tpfeq.h>
#include <c$decb.h>

DECBC_RC rc;
TPF_DECB *decb;
char     decb_name[17] = "APPLWXY         ";
  ⋮
if ( (decb = tpf_decb_create(decb_name, &rc)) != NULL )
{
        /* DECB is successfully created */
}
else
{
        /* failed to create DECB, check rc for the reason */
}
```

Related information

ALCS Application Programming Guide.

2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.85, “tpf_decb_release — Release a data event control block” on page 199.
2.86, “tpf_decb_swapblk — Swap a storage block between an ECB level and a DECB” on page 201.
2.87, “tpf_decb_validate — Validate a data event control block” on page 203.

2.84 tpf_decb_locate — Locate a data event control block

Format

```
#include <tpfeq.h>
#include <c$decb.h>

TPF_DECB *tpf_decb_locate(char *name, DECBC_RC *rc);
```

or

```
#include <tpfeq.h>
#include <c$decb.h>

TPF_DECB *tpf_decb_locate(TPF_DECB *decb, DECBC_CHAIN chain, DECBC_RC *rc);
```

name
A pointer to a 16-byte user-specified DECB name, as specified on a preceding tpf_decb_create call.

decb
A pointer to the current DECB. If NULL is specified, the first DECB, as specified by the *chain* parameter, will be returned.

chain
Indicates whether an active DECB or any DECB will be returned. Use one of the defined values:

DECBC_CHAIN_INUSE
Indicates the next active DECB after the current DECB will be returned.

DECBC_CHAIN_ANY
Indicates the next DECB, active or inactive, after the current DECB will be returned.

rc A pointer to the return code. The *rc* parameter is optional. If NULL is coded, the return code will not be set.

Description

Use the format of the tpf_decb_locate function with the *name* parameter to determine the address of a previously allocated data event control block (DECB) from it's associated name.

Alternatively, use the format of the tpf_decb_locate function with the *chain* parameter to step through each DECB associated with the ECB.

The DECB is an alternative to an ECB level, which is used to identify a DASD record that is to be read or written and to attach a storage block. The DECB fields specify the same data level and storage level information without requiring the use of an ECB level. All the same requirements and conditions that apply to the data level and storage level in the ECB also pertain to the same fields in the DECB.

The data level in the DECB provides storage for an 8-byte file address in 4x4 format.

The structure of the DECB (defined type TPF_DECB) is defined in <c$decb.h> and described in *ALCS Application Programming Guide*.

The type DECBC_RC is defined in <c$decb.h>.

The functions that support the use of a DECB (for example, file_record_ext and find_record_ext) will only accept a DECB address as a valid reference to a DECB. If an application does not maintain the address of a particular DECB and instead maintains the name of the DECB, the caller must first issue the tpf_decb_locate function to obtain the address of the desired DECB. The resulting DECB address may then be passed on to the subsequent function call.

Note: Applications that use DECBs must be compiled with the C++ compiler.

Normal return

A pointer to a DECB (defined type TPF_DECB) and the return code *rc* (defined type DECBC_RC) is set to the defined value DECBC_OK.

Error return

A NULL pointer and the return code *rc* (defined type DECBC_RC) contains the defined value DECBC_NOTFOUND if the *name* parameter specified contains a DECB name that is not valid for this ECB.

Loss of control

This function does not cause the entry to lose control.

Example

This example locates a DECB with the name "APPLWXY".

```
#include <tpfeq.h>
#include <c$decb.h>

DECBC_RC rc;
TPF_DECB *decb;
char     decb_name[17] = "APPLWXY         ";
  ⋮
if ( (decb = tpf_decb_locate(decb_name, &rc)) != NULL )
{
        /* DECB is successfully located */
}
else
{
        /* failed to locate DECB, check rc for the reason */
}
```

Related information

ALCS Application Programming Guide.

2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.85, “tpf_decb_release — Release a data event control block” on page 199.
2.86, “tpf_decb_swapblk — Swap a storage block between an ECB level and a DECB” on page 201.
2.87, “tpf_decb_validate — Validate a data event control block” on page 203.

2.85 tpf_decb_release — Release a data event control block

Format

```
#include <tpfeq.h>
#include <c$decb.h>

void tpf_decb_release(TPF_DECB *decb);
```

or

```
#include <tpfeq.h>
#include <c$decb.h>

void tpf_decb_release(char *name);
```

name
A pointer to a 16-byte user-specified DECB name, as specified on a preceding `tpf_decb_create` call.

decb
A pointer to the data event control block (DECB) to be released.

Description

Use the `tpf_decb_release` function to release a data event control block (DECB) previously created with the `tpf_decb_create` function. The DECB is detached from the ECB and the storage is available to be allocated to other DECBs.

The DECB is an alternative to an ECB level, which is used to identify a DASD record that is to be read or written and to attach a storage block. The DECB fields specify the same data level and storage level information without requiring the use of an ECB level. All the same requirements and conditions that apply to the data level and storage level in the ECB also pertain to the same fields in the DECB.

The data level in the DECB provides storage for an 8-byte file address in 4x4 format.

Note: Applications that use DECBs must be compiled with the C++ compiler.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example locates a DECB and releases it.

```
#include <tpfeq.h>
#include <c$decb.h>

DECBC_RC rc;
TPF_DECB *decb;
char     decb_name[17] = "APPLWXY         ";
  ⋮
if ( (decb = tpf_decb_locate(decb_name, &rc)) != NULL )
{
  tpf_decb_release(decb); /* release the decb that was located */
}
else
{
        /* failed to locate DECB, check rc for the reason */
}
```

Related information

2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.86, “tpf_decb_swapblk — Swap a storage block between an ECB level and a DECB” on page 201.
2.87, “tpf_decb_validate — Validate a data event control block” on page 203.

2.86 tpf_decb_swapblk — Swap a storage block between an ECB level and a DECB

Format

```
#include <tpfeq.h>
#include <c$decb.h>

void tpf_decb_swapblk(TPF_DECB *decb, enum t_lvl level);
```

level
An ECB storage level, see 2.1, “Common parameters” on page 25.

decb
A pointer to a data event control block (DECB).

Description

Use the tpf_decb_swapblk function to swap a storage block between an ECB storage level and the storage level in a data event control block (DECB). Only the storage level is swapped; the data level in both the ECB and the DECB remains unchanged.

The DECB is an alternative to an ECB level, which is used to identify a DASD record that is to be read or written and to attach a storage block. The DECB fields specify the same data level and storage level information without requiring the use of an ECB level. All the same requirements and conditions that apply to the data level and storage level in the ECB also pertain to the same fields in the DECB.

Note: Applications that use DECBs must be compiled with the C++ compiler.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example This example swaps the storage level and associated block information between ECB storage level 2 (D2) and the specified DECB.

```
#include <tpfeq.h>
#include <c$decb.h>

TPF_DECB *decb;
  ⋮
tpf_decb_swapblk(decb, D2); /* swap storage block between decb and D2 */
```

Related information

2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.85, “tpf_decb_release — Release a data event control block” on page 199.
2.87, “tpf_decb_validate — Validate a data event control block” on page 203.

2.87 tpf_decb_validate — Validate a data event control block

Format

```
#include <tpfeq.h>
#include <c$decb.h>

DECBC_RC tpf_decb_validate(TPF_DECB *decb);
```

decb
A pointer to a data event control block (DECB).

Description

Use the tpf_decb_validate function to to determine whether the specified storage address is the address of an active and valid data event control block (DECB).

The DECB is an alternative to an ECB level, which is used to identify a DASD record that is to be read or written and to attach a storage block. The DECB fields specify the same data level and storage level information without requiring the use of an ECB level. All the same requirements and conditions that apply to the data level and storage level in the ECB also pertain to the same fields in the DECB.

The structure of the DECB (defined type TPF_DECB) is defined in <c$decb.h> and described in *ALCS Application Programming Guide*.

The type DECBC_RC is defined in <c$decb.h>.

Note: Applications that use DECBs must be compiled with the C++ compiler.

Normal return

The return code (defined type DECBC_RC) is set to the defined value DECBC_OK.

Error return

The return code (defined type DECBC_RC) is set to the defined value DECBC_NOTVALID.

Loss of control

This function does not cause the entry to lose control.

Example

This example validates a DECB address.

```
#include <tpfeq.h>
#include <c$decb.h>

TPF_DECB *decb;
  ⋮
if ( tpf_decb_validate(decb) == DECBC_OK )
{
    /* decb is valid */
}
else
{
    /* DECB is not valid */
}
```

Related information

ALCS Application Programming Guide.

2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.85, “tpf_decb_release — Release a data event control block” on page 199.
2.86, “tpf_decb_swapblk — Swap a storage block between an ECB level and a DECB” on page 201.

2.88 tpf_fac8c — Compute an online 8-byte file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

int tpf_fac8c(TPF_FAC8 *parm);
```

parm
A pointer to a parameter block described by the structure defined as type TPF_FAC8. Fields in the input area of this block must be initialized by the caller before the function is called. Output area fields will be filled in by the function service routine.

Description

Use the tpf_fac8c function to compute an 8-byte file address based on the input record type and 8-byte ordinal number. The service is similar to calling the face or facs functions.

The data level in a data event control block (DECB) provides storage for the 8-byte file address.

The structure of the parameter block (defined type TPF_FAC8) is defined in <tpfio.h>. See 1.22, "<tpfio.h>" on page 22.

Before calling tpf_fac8c you must set the record ordinal number (ifacord) and the fixed-file record type (value (ifacrnb) or symbol (ifacrec)) in the parameter block. The symbolic record type name, if specified, must be 8 bytes, left justified and padded with space characters.

Before calling tpf_fac8c you must set field (ifactyp) in the parameter block to indicate whether the record type specified is a symbol (IFAC8FCS) or a value (IFAC8FCE).

Note: Applications that use 8-byte file addresses must be compiled with the C++ compiler.

TPF Compatibility

The 8-byte file address support is required for compatibility with TPF. In ALCS, the tpf_fac8c function returns an 8-byte file address in 4x4 format, where the high-order 4 bytes contain an indicator (a full-word of zeros) that classifies it as a valid 4x4 format file address and the low-order 4 bytes contain an ALCS band format file address.

Normal return

An integer containing the defined value TPF_FAC8_NRM and the following are stored in fields in the output area of the parameter block. The ifacret contains the return code (the same as the return value from tpf_fac8c), ifacmax contains the maximum ordinal number for the requested record type and ifacadr contains the 8-byte file address.

Error return

An integer containing one of the following defined values:

TPF_FAC8_TYP
: Call type invalid or record type invalid.

TPF_FAC8_ORD
: Record ordinal invalid.

Loss of control

This function does not cause the entry to lose control.

Example

This example generates the file address for ordinal 198 in fixed-file record type #WAARI and stores it in the output area of the TPF_FAC8 block.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_FAC8 parm;
  ⋮
parm.ifacord = 198;        /* set ordinal */
                           /* set rec type padded with blanks */
memcpy(parm.ifacrec, "#WAARI  ", sizeof(parm.ifacrec));
parm.ifactyp = IFAC8FCS;   /* set call type - rec type is symbol */
tpf_fac8c(&parm);          /* call tpf_fac8c to calculate file address */

if (parm.ifacret != TPF_FAC8_NRM)
{
  /* error return */
}
else
{
  /* normal return */
}
```

Related information

2.23, “face — Compute the file address of a fixed file record from a record type value” on page 63.
2.24, “facs — Compute the file address of a fixed file record from a record type symbol” on page 65.
2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.85, “tpf_decb_release — Release a data event control block” on page 199.
1.22, “<tpfio.h>” on page 22.

2.89 tpf_fa4x4c — Convert a file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

int tpf_fa4x4c(TPF_FACONV *action, unsigned long *fa4, TPF_FA8 *fa8);
```

or format for use in ALCS only:

```
#include <tpfeq.h>
#include <tpfio.h>

int tpf_fa4x4c(TPF_FACONV *action, unsigned int *fa4, TPF_FA8 *fa8);
```

action
The type of conversion to be performed. Use one of the values:

FACONV_4TO8
Converts a 4-byte file address to an 8-byte file address in 4x4 format.

FACONV_8TO4
Converts an 8-byte file address in 4x4 format to a 4-byte file address.

fa4
A pointer to a 4-byte file address. This parameter is the input if *action* is FACONV_4TO8. This parameter is the output if *action* is FACONV_8TO4.

fa8
A pointer to an 8-byte file address. This parameter is the input if *action* is FACONV_8TO4. This parameter is the output if *action* is FACONV_4TO8.

Description

Use the tpf_fa4x4c function to:

- 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.

4x4 format is an 8-byte file address with a high-order 4-byte indicator (that contains zeros) and a low-order 4 bytes that contains an ALCS band format file address.

The data level in a data event control block (DECB) provides storage for an 8-byte file address. Use the tpf_fa4x4c function to convert an existing 4-byte file address to an 8-byte file address and place it in a DECB data level. Use the tpf_fa4x4c function to convert an 8-byte file address obtained using the data level in a DECB to a 4-byte file address for subsequent use with an ECB data level.

The type TPF_FA8 is defined in <c$decb.h> (#included by <tpfio.h>).

Note: Applications that use 8-byte file addresses must be compiled with the C++ compiler.

Normal return
A non-zero integer.

Error return
An integer value of zero. This can occur if FACONV_8TO4 is specified and the input 8-byte file address is not in 4x4 format, or if the specified action is not valid.

Loss of control
This function does not cause the entry to lose control.

Example
This example converts a 4-byte file address to an 8-byte file address and converts an 8-byte file address to a 4-byte file address.

```
#include <tpfeq.h>
#include <tpfio.h>

unsigned long fa4;
TPF_FA8       fa8;
  ⋮
tpf_fa4x4c(FACONV_4TO8, &fa4, &fa8); /* convert 4-byte fa to 8-byte */
  ⋮
tpf_fa4x4c(FACONV_8TO4, &fa4, &fa8); /* convert 8-byte fa to 4-byte */
```

Related information
2.83, “tpf_decb_create — Create a data event control block” on page 195.
2.84, “tpf_decb_locate — Locate a data event control block” on page 197.
2.85, “tpf_decb_release — Release a data event control block” on page 199.

2.90 tpf_rcrfc — Convert a file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void tpf_rcrfc(enum t_lvl *level);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void tpf_rcrfc(TPF_DECB *decb);
```

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This indicates the data level containing the pool-file address to be released and the storage level containing the address of the storage block to be released.

decb
A pointer to a data event control block (DECB).

Description

Use the tpf_rcrfc function to release a pool-file address and associated storage block.

tpf_rcrfc is the same as relcc followed by relfc. See the **Description** sections for relcc and relfc.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example releases a storage block and pool-file address from an ECB level and a DECB.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_DECB *decb;
  ⋮
tpf_rcrfc(D6);
  ⋮
tpf_rcrfc(decb);
```

Related information

2.61, “relcc — Release a storage block” on page 148.
2.62, “relfc — Release a pool-file record address” on page 150.

2.91 tpf_STCK — Store clock

Format

```
#include <tpfeq.h>
#include <tpfapi.h>

int tpf_STCK(tpf_TOD_type *tod);
```

tod
A pointer to the 8-byte area that is to receive the value of the TOD clock. On return, ALCS sets this area to the current value of the TOD clock. The pointer is always of type tpf_TOD_type. The type specifier tpf_TOD_type is declared in <tpfapi.h> and is a union containing the following fields:

stck_value
Double field containing bits 0-63 of the TOD clock.

bits
Structure containing the following fields:

hi Unsigned long field containing bits 0-31 of the TOD clock.

lo Unsigned long field containing bits 32-63 of the TOD clock.

Description

Use the tpf_STCK function to return the current value of bits 0-63 of the S/390 time-of-day (TOD) clock obtained with the STCK instruction.

Normal return

The function returns STCK_OK if the clock is in the set state. The TOD clock value returned is a valid time-of-day indication.

Error return

The function returns STCK_ERROR if the clock is in the not-set state, error state, stopped state or not-operational state. The TOD clock value returned is not a valid time-of-day indication.

Loss of control

This function does not cause the entry to lose control.

Example

This example shows the use of tpf_STCK to get the current value of the TOD clock. If normal return, the value is displayed. If error return, an error message is displayed.

```
#include <tpfeq.h>
#include <tpfapi.h>

tpf_TOD_type *tod_time;

if ( tpf_STCK(tod_time) == STCK_OK )   /* normal return */
  printf("TOD clock value is %08lx%08lx", tod_time->bits.hi,
                                          tod_time->bits.lo );

else                                   /* error return  */
  puts("tpf_STCK error");
```

Related information

z/Architecture Principles of Operation – STCK instruction.
C/C++ Run-Time Library Reference – `time()` function.
2.79, “`timec` — Get time and date” on page 187.

2.92 tprdc — Read a record from a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void tprdc(char *name, enum t_lvl level, enum t_blktype size);
```

name
Pointer to the 3-character name of the sequential file that you want to read from.

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This is the storage level on which ALCS attaches the storage block containing the record image. (This storage level must not have an attached storage block when you call tprdc.)

size
This is a block size code. Use one of the values: L1, L2, ... L8 defined in your installation.

Description

Use the tprdc function to read a record from a general sequential file into a storage block of the specified size. tprdc attaches the block at the level you specify in the *level* parameter.

The general sequential file must have been opened (using topnc) for input and must be assigned (tasnc) to the entry.

The function reads a record from the current position in the sequential file.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

You must follow the function call (at some stage) with a waitc call, or with a call to a function that has an implied waitc. The return value of the waitc lets you check whether the preceding I/O operations, including this read, were successful.

Example

This example reads a size L2 record from the sequential file VPH onto level 2 (D2). If an I/O error occurs, it produces a dump, number 001234.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tprdc("VPH",D2,L2);
if (waitc())
  {
    errno = 0x1234;
    perror("ERROR READING VPH TAPE");
    abort();
  }
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.77, “tclsc — Close a general sequential file” on page 184.
2.80, “topnc — Open a general sequential file” on page 191.
2.93, “trsvc — Reserve a general sequential file” on page 215.
2.94, “twrtc — Write a record to a general sequential file” on page 216.
2.97, “waitc — Wait for all outstanding I/O requests to complete” on page 222.

2.93 trsvc — Reserve a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void trsvc(char *name);
```

name
Pointer to the 3-character name of the sequential file to be reserved.

Description

Use the trsvc function to reserve (unassign) a general sequential file. trsvc does not close the file, another (or the same) entry can assign the file using the tasnc function.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example reserves the sequential file VPH for use by other entries.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
trsvc("VPH");
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.77, “tclsc — Close a general sequential file” on page 184.
2.80, “topnc — Open a general sequential file” on page 191.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.94, “twrtc — Write a record to a general sequential file” on page 216.

2.94 twrtc — Write a record to a general sequential file

Format

```
#include <tpfeq.h>
#include <tpftape.h>

void twrtc(char *name, enum t_lvl level);
```

name
Pointer to the 3-character name of the sequential file you want to write to.

level
An ECB storage level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. The storage block on this level contains the image of the record that ALCS writes to the sequential file. (The block size must be the same as the record size in the sequential file.)

Description

Use the twrtc function to write a record to a general sequential file. twrtc writes the record from the storage block that is attached at the specified level. ALCS then detaches the block from the level. (The entry must not refer to the detached block after calling this function.)

The specified general sequential file must be open and assigned to the current entry.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function can cause the entry to lose control.

The function does not require a subsequent waitc function. The entry cannot check if the write was successful.

Example

This example writes the storage block on level 3 (D3) to the sequential file VPH. On return, the block is no longer available to the application program.

```
#include <tpfeq.h>
#include <tpftape.h>

  ⋮
tasnc("VPH");
twrtc("VPH",D3);
trsvc("VPH");
```

Related information

2.76, “tasnc — Assign a general sequential file” on page 183.
2.77, “tclsc — Close a general sequential file” on page 184.
2.80, “topnc — Open a general sequential file” on page 191.
2.92, “tprdc — Read a record from a general sequential file” on page 213.
2.93, “trsvc — Reserve a general sequential file” on page 215.

2.95 unfrc — Unhold a file address

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void unfrc(enum t_lvl level [,GDS]);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This data level contains the file address of the record you want to unhold. The file address must have been held by the calling entry.

GDS
See 2.1, “Common parameters” on page 25.

Description

Use the unfrc function to unhold a file address that the entry has previously held with a finhc, or fiwhc function call, or a find_record call with a HOLD value of the *type* parameter.

The function does not initiate any I/O and does not check or modify the storage level.

Normal return

Void.

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Example

This example finds (with hold) an application message block on level 6 and then unholds the file address.

```
#include <tpfeq.h>
#include <tpfio.h>
#include <c$am0sg.h>

struct am0sg *prime, *chain;          /* Pointers to message blocks */
  ⋮
prime = ecbptr()->ce1cr1;             /* Base prime message block   */
                                      /* and read 1st chain w/hold  */
chain = find_record(D6,(unsigned int *)&(prime->am0fch),"OM",'\0',HOLD);
unfrc(D6);                            /* Now remove chain address from
                                         the record hold status     */
```

Related information

2.35, “find_record — Read a DASD record” on page 90.
2.37, “finhc — Read a DASD record and hold the file address” on page 97.
2.41, “fiwhc — Read a DASD record, hold the file address, and wait for I/O completion” on page 105.

2.96 unfrc_ext — Unhold a file address, with extended options

Format

```
#include <tpfeq.h>
#include <tpfio.h>

void unfrc_ext(enum t_lvl *level, unsigned int ext_find);
```

or

```
#include <tpfeq.h>
#include <tpfio.h>

void unfrc_ext(TPF_DECB *decb, unsigned int ext_find);
```

level
An ECB data level (D0, D1, ..., DF), see 2.1, “Common parameters” on page 25. This data level contains the file address of the record to be unheld. The file address must have been held by the calling entry.

decb
A pointer to a data event control block (DECB). This DECB contains the file address of the record to be unheld. The file address must have been held by the calling entry.

ext_find
ALCS ignores any values you code in this parameter (provided for compatibility with TPF). See 2.1, “Common parameters” on page 25.

Description

Use the `unfrc_ext` function to unhold a file address that the entry has previously held with a `finhc` , or `fiwhc` function call, or a `find_record` call with a HOLD value of the *type* parameter. The file address is contained in the ECB data level specified in the *level* parameter, or the data level of the DECB specified in the *decb* parameter.

The function does not initiate any I/O and does not check or modify the storage level.

Note: Applications that call this function using a DECB address instead of an ECB level must be compiled with the C++ compiler because this function has been overloaded.

Normal return

Void

Error return

Not applicable.

Loss of control

This function does not cause the entry to lose control.

Examples

This example unholds the file address on level D7.

```
#include <tpfeq.h>
#include <tpfio.h>
  ⋮
unfrc_ext(D7, FIND_GDS);
```

This example unholds the file address in a decb.

```
#include <tpfeq.h>
#include <tpfio.h>

TPF_DECB *decb;
  ⋮
unfrc_ext(decb, FIND_DEFEXT);
```

Related information

2.35, "find_record — Read a DASD record" on page 90.
2.37, "finhc — Read a DASD record and hold the file address" on page 97.
2.41, "fiwhc — Read a DASD record, hold the file address, and wait for I/O completion" on page 105.

2.97 waitc — Wait for all outstanding I/O requests to complete

Format

```
#include <tpfeq.h>
#include <tpfio.h>

int waitc(void);
```

Description

Use the waitc function to suspend the issuing entry until all I/O requests for this entry which either require a subsequent waitc call, or contain an implied waitc, are complete.

Do not use waitc to prevent an application loop timeout. To make sure that the entry loses control. use dlayc or defrc instead. (If I/O is in progress, use waitc to ensure that the I/O is complete on return from dlayc or defrc).

For more information about waitc, see *ALCS Application Programming Guide*.

Normal return

Integer value of zero.

Error return

The result of carrying out a logical OR operation on all 16 error fields (ebcsd*n*) in the ECB. If an I/O error has occurred at any of the 16 data levels, this error return is nonzero.

Loss of control

This function causes the entry to lose control, unless there are no outstanding I/O requests, when waitc has no effect.

Example

This example uses waitc to ensure that all outstanding I/O has completed. If an I/O error occurs, it produces a dump number 001234 and terminates abnormally.

```
#include <tpfeq.h>
#include <tpfio.h>

  ⋮
if(waitc())
   {
    errno = 0x1234;
    perror("I/O error occurred");
    abort();
   }
```

Related information

ALCS Application Programming Guide.
2.29, “filnc — Write a DASD record and retain the attached storage block” on page 78.
2.33, “findc — Read a DASD record” on page 86.
2.37, “finhc — Read a DASD record and hold the file address” on page 97.
2.92, “tprdc — Read a record from a general sequential file” on page 213.

2.98 wtopc — Output message services

Format

```
#include <tpfeq.h>
#include <tpfapi.h>
int wtopc(const char *text_ptr, long int rout,
          enum t_wtopc_chain chain, const void *header_ptr);
```

text_ptr
Pointer to the output message text. The text is a character string of up to 255 characters terminated by a NULL.

rout
Destination for the message. The addition of one or more of:

WTOPC_EBROUT
CRI in ebrout (this is normally the terminal that originated the input message your program is processing)

WTOPC_RO
RO CRAS

WTOPC_PRC
Prime CRAS

WTOPC_TAPE
Tape console

WTOPC_DASD
DASD console

WTOPC_COMM
Communication console

WTOPC_AUDT
Audit console

NULL
CRI in ebrout (same as WTOPC_EBROUT)

If you specify any routing destination other than WTOPC_EBROUT, ALCS uses RO CRAS as a destination.

chain
Chaining for the text in this wtopc call. Chaining allows you to send a multiline message with a sequence of wtopc calls. One of:

WTOPC_NO_CHAIN
This is a complete message.

WTOPC_CHAIN
This is one line (not the last) of a multiline message.

WTOPC_CHAIN_END
This is the last line of a multiline message.

WTOPC_NO_PAGE
Identical to WTOPC_NO_CHAIN.

WTOPC_PAGE
Identical to WTOPC_CHAIN.

WTOPC_PAGE_END
Identical to WTOPC_CHAIN_END.

header_ptr
Pointer to a structure that contains details of the message header, or NULL to use the default message header, or WTOPC_NO_HEADER if the message does not have a header. The structure is called `wtopc_header` and is defined in header `<tpfapi.h>`. It contains the following fields:

`wtopc_prefix_pointer`
`char` pointer to the four character message prefix, or NULL to use the name of your program as the message prefix.

`wtopc_number`
`short int` message number, in the range 0-9999.

`wtopc_letter`
`char` message severity code, or `'\0'` to use the default severity code `'I'`.

`wtopc_time`
Message time. One of:

WTOPC_SYS_TIME
Include the system (MVS) time in the message header.

WTOPC_SUBSYS_TIME
Include the ALCS time in the message header.

WTOPC_NO_TIME
Do not include the time in the message header.

0
Include the ALCS time in the message header (same as WTOPC_SUBSYS_TIME).

If you specify *header_ptr* as NULL, ALCS constructs the default header using the individual defaults shown above, and with a message number of `'0001'`.

Description

Use the `wtopc` function to send an output message either to the originator of the input message your program is processing, or to RO CRAS, or both.

Your output message can optionally include a header containing a four-character prefix, a decimal message number, a single-character severity code, and optionally a time.

You can use a sequence of `wtopc` calls to build and send a single multiline message. Each `wtopc` call with WTOPC_CHAIN adds one line to the message; the final `wtopc` call with WTOPC_CHAIN_END adds the last line and sends the message.

ALCS provides two additional functions that can simplify coding `wtopc` calls, see 2.99, “`wtopc_insert_header` — Save header for `wtopc`” on page 227 and 2.100, “`wtopc_routing_list` — Save routing list for `wtopc`” on page 229. ALCS also provides a similar but simpler function that you may be able to use instead of `wtopc`, see 2.101, “`wtopc_text` — Send output message” on page 231.

Normal return
Zero.

Error return
Not applicable.

Loss of control
This function can cause the entry to lose control.

Examples
The following example sends a single line output message to the originator of the input message the program is processing (the originating terminal address is in `ebrout`). The example uses `wtopc_insert_header` to construct the message header.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
struct wtopc_header header_buffer;
⋮
wtopc_insert_header(&header_buffer, "C000", 2, 'A', WTOPC_SYS_TIME);
wtopc("msg1", WTOPC_EBROUT, WTOPC_NO_CHAIN, &header_buffer);
⋮
```

The following example sends a two-line message to RO CRAS. It uses the default header.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
wtopc("msg2", WTOPC_RO, WTOPC_CHAIN, NULL);
wtopc("msg3", WTOPC_RO, WTOPC_CHAIN_END, NULL);
⋮
```

The following example sends a single line output message to the originator of the input message the program is processing, and a copy of the message to RO CRAS. It uses `wtopc_routing_list` to construct the routing list and `sprintf` to construct the text. The message does not have a header.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
long int rout_buffer;
char      message_buffer[100];
char *    s ="msg";
⋮
sprintf(message_buffer, "%s %d\n", s, 4);
wtopc_routing_list(rout_buffer, WTOPC_EBROUT + WTOPC_RO);
wtopc(message_buffer, rout_buffer, WTOPC_NO_CHAIN, WTOPC_NO_HEADER);
⋮
```

Related information

2.99, “wtopc_insert_header – Save header for wtopc” on page 227
2.100, “wtopc_routing_list – Save routing list for wtopc” on page 229
2.101, “wtopc_text – Send output message” on page 231

2.99 wtopc_insert_header — Save header for wtopc

Format

```
#include <tpfeq.h>
#include <tpfapi.h>
void wtopc_insert_header(char *buffer_ptr, char *prefix_ptr,
                         short int number, char letter,
                         enum t_wtopc_time time);
```

buffer_ptr
Pointer to a structure where wtopc_insert_header stores details of the message header. The structure is called wtopc_header and is defined in header <tpfapi.h>.

prefix_ptr
char pointer to the four character message prefix, or NULL to use the name of your program as the message prefix.

number
short int message number, in the range 0-9999.

letter
char message severity code, or '\0' to use the default severity code 'I'. IBM recommends that you use a severity code of 'I', 'W', or 'E'. You can use names for these severity codes as shown in Figure 3.

Figure 3. wtopc message severity codes

You can use...	...for...	...which means
WTOPC_INFO	I	Information message
WTOPC_WARNING	W	Attention message
WTOPC_ERROR	E	Error message

time
Message time. One of:

WTOPC_SYS_TIME
: Include the system (MVS) time in the message header.

WTOPC_SUBSYS_TIME
: Include the ALCS time in the message header.

WTOPC_NO_TIME
: Do not include the time in the message header.

0
: Include the ALCS time in the message header (same as WTOPC_SUBSYS_TIME).

Description

You can use this function to build a wtopc_header structure which defines the header format for a wtopc message. You can pass the structure that wtopc_insert_header builds as a parameter in your wtopc call.

Normal return
Void.

Error return
Not applicable.

Examples
The following example sets up a header for `wtopc` to use.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
struct wtopc_header header_buffer;
⋮
wtopc_insert_header(&header_buffer, "C000", 2, 'A', WTOPC_SYS_TIME);
⋮
```

The following example sets up a header for `wtopc` to use. It defualts the message prefix to the calling program name.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
struct wtopc_header header_buffer;
⋮
wtopc_insert_header(&header_buffer, NULL, 2, 'A', WTOPC_SYS_TIME);
⋮
```

Related information
2.98, “`wtopc` — Output message services” on page 223
2.100, “`wtopc_routing_list` — Save routing list for `wtopc`” on page 229
2.101, “`wtopc_text` — Send output message” on page 231

2.100 wtopc_routing_list — Save routing list for wtopc

Format

```
#include <tpfeq.h>
#include <tpfapi.h>
void wtopc_routing_list(long int rout, long int codes);
```

rout
long int where wtopc_routing_list builds the routing list information.

codes
Destination for the message. The addition of one or more of:

WTOPC_EBROUT
CRI in ebrout (this is normally the terminal that originated the input message your program is processing)

WTOPC_RO
RO CRAS

WTOPC_PRC
Prime CRAS

WTOPC_TAPE
Tape console

WTOPC_DASD
DASD console

WTOPC_COMM
Communication console

WTOPC_AUDT
Audit console

NULL
CRI in ebrout (same as WTOPC_EBROUT)

If you specify any routing destination other than WTOPC_EBROUT, ALCS uses RO CRAS as a destination.

Description

You can use this function to build a list of destinations for a wtopc message. You can pass the long int that wtopc_routing_list builds as a parameter in your wtopc call.

Normal return

Void.

Error return

Not applicable.

Examples

The following example sets up a routing list for `wtopc`. The destinations are the originator of the input message the program is processing, and RO CRAS.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
long int  rout_list;
⋮
wtopc_routing_list(rout_list, WTOPC_EBROUT + WTOPC_RO);
⋮
```

Related information

2.98, "`wtopc` — Output message services" on page 223
2.99, "`wtopc_insert_header` — Save header for `wtopc`" on page 227
2.101, "`wtopc_text` — Send output message" on page 231

2.101 wtopc_text — Send output message

Format

```
#include <tpfeq.h>
#include <tpfapi.h>
void wtopc_text(const char *text_ptr);
```

text_ptr
Pointer to the output message text. The text is a character string of up to 255 characters terminated by a NULL.

Description

Use the wtopc_text function to send a single-line output message to the originator of the input message your program is processing.

Your output message includes a header containing

A four-character prefix (the name of your program)
A decimal message number ('0001')
A single-character severity code ('I')
The ALCS time.

Normal return

Void.

Error return

Not applicable.

Example

The following example sends a message to the originator of the input message your program is processing.

```
#include <tpfeq.h>
#include <tpfapi.h>
⋮
wtopc_text("msg1");
⋮
```

Related information

2.98, "wtopc — Output message services" on page 223
2.99, "wtopc_insert_header — Save header for wtopc" on page 227
2.100, "wtopc_routing_list — Save routing list for wtopc" on page 229

Appendix A. C library functions available under ISO-C, ALCS, and TPF

Figure 4 (Page 1 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
■ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
abort	Abnormally stops a program	√	■	■
abs	Calculates the absolute value of an integer	√	√	√
acos	Calculates the arc cosine	√	√	√
asctime	Converts time stored as a structure to a character string in store	√	√	√
asin	Calculates the arc sine	√	√	√
assert	Prints diagnostic message	√	■	■
atan	Calculates the arc tangent of x	√	√	√
atan2	Calculates the arc tangent of x/y	√	√	√
atawait	Waits for one or more asynchronous APPC/MVS calls to complete		X	
atexit	Records program termination routine	√	√	
atof	Converts a character string to float	√	√	√
atoi	Converts a character string to integer	√	√	√
atol	Converts a character string to long integer	√	√	√
attac	Attaches a previously detached storage block		X	X
attac_ext	Attaches a previously detached storage block		X	X
attac_id	Attaches a previously detached storage block		X	X
bessel	Bessel functions	√	√	
bsearch	Performs a binary search of a sorted array	√	√	√ ISO-C
calloc	Reserves storage space for an array and initializes the values of all elements to 0	√	√	√
ceil	Calculates the double value representing the smallest integer that is greater than or equal to a number	√	√	√
cifrc	Cipher program interface			X
cinfc	Extracts data from monitor table			X
cinfc_fast	Extracts data from monitor table, fast			X
clearerr	Resets error indicators	√	√	
clock	Determines processor time	√	√	√ ISO-C

Figure 4 (Page 2 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
■ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
comic	Gets communication resource information		X	
corhc	Defines and holds a resource		X	X
coruc	Unholds a resource		X	X
cos	Calculates the cosine	√	√	√
cosh	Calculates the hyperbolic cosine	√	√	√
credc	Creates an entry for deferred scheduling		X	X
creec	Creates an entry with an attached storage block		X	X
cremc	Creates an entry for immediate scheduling		X	X
cretc	Creates an entry for scheduling after a time delay		X	X
crexc	Creates an entry for deferred scheduling		X	X
crusa	Releases storage blocks from specified levels		X	X
ctime	Converts time stored as a long value to a character string	√	√	√
defrc	Defers processing of current entry		X	X
detac	Detaches a storage block from the ECB or DECB		X	X
detac_ext	Detaches a storage block from the ECB or DECB		X	X
detac_id	Detaches a storage block from the ECB or DECB		X	X
difftime	Computes the difference between two times	√	√	√
div	Calculates the quotient and remainder	√	√	√
dlayc	Delays processing of the current entry		X	X
ecbptr	Accesses the entry control block		X	X
entdc	Enters a program with no return allowed to any program		X	X
entrc	Enters a program with a return expected to the calling program		X	
erf	Calculates the error function	√	√	
erfc	Calculates the error function for large numbers	√	√	
exit	Normally ends a program	√	■	■
exp	Calculates an exponential function	√	√	√
fabs	Calculates the absolute value of a floating point number	√	√	√
face	Computes an online file address from a record type number		X	X
facs	Computes an online file address from a record type symbol		X	X
fclose	Closes a specified stream	√	M	

Figure 4 (Page 3 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
▪ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
feof	Tests end-of-file indicator for stream input	√	M	
ferror	Tests the error indicator for a specific stream	√	M	
fflush	Causes the system to write the contents of a buffer to a file	√	M	
fgetc	Reads a character from a specified input stream	√	M	
fgetpos	Gets the current position of the file pointer	√	M	
fgets	Reads a string from a specified input stream	√	M	
filec	Writes a DASD record		X	X
filec_ext	Writes a record, with extended options		X	X
file_record	Writes a DASD record		X	X
file_record_ext	Writes a record, extended		X	X
filnc	Writes a DASD record and retains the attached storage block		X	X
filnc_ext	Writes a DASD record and retains the attached storage block, with extended options		X	X
filuc	Writes a DASD record and unholds the file address		X	X
filuc_ext	Writes a DASD record and unholds the file address, with extended options		X	X
findc	Reads a DASD record		X	X
find_record	Reads a DASD record		X	X
find_record_ext	Reads a DASD record, with extended options		X	X
finhc	Raeds a DASD record and holds the file address		X	X
finhc_ext	Reads a DASD record and holds the file address, with extended options		X	X
finwc	Reads a DASD record and waits for I/O completion		X	X
finwc_ext	Reads a DASD record and waits for I/O completion, with extended options		X	X
fiwhc	Reads a DASD record, holds the file address, and waits for I/O completion		X	X
fiwhc_ext	Reads a DASD record, holds the file address, and waits for I/O completion, with extended options,		X	X
flipc	Exchanges the content of two storage and data levels		X	X
floor	Calculates the floating point value representing the largest integer less than or equal to a number	√	√	√
fmod	Calculates the floating point remainder of one argument divided by another	√	√	√
fopen	Opens a specified stream	√	M	

Figure 4 (Page 4 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
■ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
fprintf	Formats and prints characters to the output stream	√	M	
fputc	Prints a character to a specified output stream	√	M	
fputs	Prints a string to a specified output stream	√	M	
fread	Reads items from a specified input stream	√	M	
free	Frees storage blocks	√	√	√
freopen	Closes a file and reassigns a stream	√	M	
frexp	Separates a floating point number into its mantissa and exponent	√	√	√
fscanf	Reads data from a stream into locations given by arguments	√	M	
fseek	Moves the file pointer to a new location	√	M	
fsetpos	Moves the file pointer to a new location	√	M	
ftell	Gets the current position of the file pointer	√	M	
fwrite	Writes items to a specified output stream	√	M	
gamma	Calculates the gamma function	√	√	
gdsnc	Opens or closes a general data set		X	X
gdsrc	Computes a general data set record file address		X	X
getc	Reads a character from a specified input stream	√	M	
getcc	Gets a storage block		X	X
getchar	Reads a character from stdin	√	√	
getenv	Searches environment variables for a specified variable	√	√	
getfc	Gets a pool file record file address		X	X
gets	Reads a line	√	√	■
glob	Addresses an application global field or record		X	X
global	Operates on an application global field		X	X
gmtime	Converts time to a structure	√	√	√
hypot	Calculates the hypotenuse	√	√	
inqrc	Converts communication resource identifiers and names			X
isalnum	Tests for alphanumeric characters	√	√	√
isalpha	Tests for alphabetic characters	√	√	√
iscntrl	Tests for control characters	√	√	√
isdigit	Tests for decimal digits	√	√	√
isgraph	Tests for printable characters excluding the space	√	√	√

Figure 4 (Page 5 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
▪ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
`islower`	Tests for lowercase letters	√	√	√
`isprint`	Tests for printable characters including the space	√	√	√
`ispunct`	Tests for printable characters excluding the space	√	√	√
`isspace`	Tests for whitespace characters	√	√	√
`isupper`	Tests for uppercase letters	√	√	√
`isxdigit`	Tests for hexadecimal digits	√	√	√
`labs`	Calculates the absolute value of a long integer	√	√	√
`ldexp`	Multiplies a floating point number by an integral power of 2	√	√	√
`ldiv`	Calculates the quotient and remainder	√	√	√
`levtest`	Tests a storage level		X	X
`localeconv`	Formats numeric quantities in struc lconv according to the current locale	√	√	√
`localtime`	Converts time to a structure	√	√	√
`lodic`	Extracts system load information		X	
`log`	Calculates the natural logarithm	√	√	√
`log10`	Calculates the base 10 logarithm	√	√	√
`longc`	Sets maximum entry life		X	X
`longjmp`	Restores a stack environment	√	√	
`malloc`	Allocates storage	√	√	√
`mblen`	Determines the length of a string	√	√	√ ISO-C
`mbstowcs`	Converts multibyte string to codes	√	√	√ ISO-C
`mbtowc`	Converts multibyte characters to `wchar_t` characters	√	√	√ ISO-C
`memchr`	Searches a buffer for the first occurrence of a given character	√	√	√
`memcmp`	Compares two buffers	√	√	√
`memcpy`	Copies a string of bytes	√	√	√
`memmove`	Moves a string of bytes	√	√	√
`memset`	Sets a string of bytes to a given value	√	√	√
`mktime`	Converts local time into calender time	√	√	√
`modec`	Sets current addressing mode		X	X
`modf`	Calculates the integer remainder of one argument divided by another	√	√	√

Figure 4 (Page 6 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
■ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
mqawait	Waits for one or more assynchronous MQI calls to complete		X	
perror	Issues system error with message	√	■	■ Not in ISO-C
pow	Calculates the value of an argument raised to a power	√	√	√
printf	Sends formatted message to terminal	√	√	■
putc	Prints a character to a specified output stream	√	M	
putchar	Prints a character to stdout	√	√	
puts	Writes a string to stdout	√	√	■
qsort	Performs a search on an array of elements	√	√	√ ISO-C
raisa	Computes the file address of a general file record		X	
raise	Sends signal	√	X	X
rand	Returns a pseudo-random integer	√	√	√
rcunc	Releases a storage block and unholds a file address		X	X
realloc	Changes storage size allocated for an object	√	√	√
rehka	Rehooks storage address			X
relcc	Releases a storage block		X	X
relfc	Releases a pool file record file address		X	X
remove	Deletes a specified file	√	M	
rename	Renames a specified file	√	M	
rewind	Repositions the file pointer to the beginning of the file	√	M	
rlcha	Releases a chain of pool file record file addresses		X	X
ronic	Returns information about a DASD record or record type		X	
routc	Routes a message to a terminal or application		X	X
scanf	Scans input for variables	√	√	■
selec	Selects a thread application interface			X
serrc_op	Requests an error dump		X	X
serrc_op_ext	Requests an error dump, with extended options		X	X
serrc_op_slt	Requests an error dump, with storage list		X	X
setbuf	Allows control of buffering	√		
setjmp	Saves a stack environment	√	√	

Figure 4 (Page 7 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
▪ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
setlocale	Changes or queries locale	√	√	√
setvbuf	Controls buffering and buffer size for a specified stream	√		
signal	Allows handling of an interrupt signal from the operating system	√	√	
sin	Calculates the sine	√	√	√
sinh	Calculates the hyperbolic sine	√	√	√
slimc	Sets or removes processing limits for the entry		X	
snapc	Requests an error dump		X	X
sonic	Get information about a file address		X	X
sprintf	Formats and stores characters in a buffer	√	√	√
sqrt	Calculates the square root of a number	√	√	√
srand	Sets the starting point for pseudo-random numbers	√	√	√
sscanf	Reads data from a buffer into locations given by arguments	√	√	√
strcat	Concatenates two strings	√	√	√
strchr	Locates the first occurrence of a specified character in a string	√	√	√
strcmp	Compares the value of two strings	√	√	√
strcoll	Compares the locale-defined value of two strings	√	√	√
strcpy	Copies one string to another	√	√	√
strcspn	Finds the length of the first substring in a string of characters not in a second string	√	√	√
strerror	Sets pointer to system error message	√	√	
strftime	Multibyte time conversion	√	√	√
strlen	Calculates the length of a string	√	√	√
strncat	Concatenates a part of a string to another string	√	√	√
strncmp	Compares parts of two strings	√	√	√
strncpy	Copies part of one string to another	√	√	√
strpbrk	Locates specified characters in a string	√	√	√
strrchr	Locates the last occurrence of a set of characters in a string	√	√	√
strspn	Locates the first character in a string that is not part of a specified set of characters	√	√	√
strstr	Locates the first occurrence of a string in another string	√	√	√
strtod	Converts a character string to double	√	√	√

Figure 4 (Page 8 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
■ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
strtok	Locates a specified token in a string	√	√	√
strtol	Converts a character string to long integer	√	√	√
strtold	Converts a character string to a double integer	√	√	√
strtoul	Converts a character string to an unsigned long integer	√	√	√
strxfrm	Transforms strings according to locale	√	√	√
system	Passes a string to the command interpreter	√		
tan	Calculates the tangent	√	√	√
tanh	Calculates the hyperbolic tangent	√	√	√
tape_close	Closes a general sequential file		X	X
tape_cntl	Tape control			X
tape_open	Opens a general sequential file		X	X
tape_read	Reads a record from a general sequential file		X	X
tape_write	Writes a record to a general sequential file		X	X
tasnc	Assigns a general sequential file		X	X
tbspc	Backspaces a tape and waits			X
tclsc	Closes a general sequential file		X	X
tconfirmed	Sends a confirmation reply			
tdspc	Gets information about any type of sequential file		X	X
time	Returns the time in seconds	√	√	√
timec	Gets time and date		X	
tmpfile	Creates a temporary file and returns a pointer to that file	√		
tmpnam	Produces a temporary file name	√		
tolower	Converts a character to lowercase	√	√	√
topnc	Opens a general sequential file		X	X
toupper	Converts a character to uppercase	√	√	√
tourc	Writes a storage block to a real-time sequential file		X	X
toutc	Writes a record to a real-time sequential file		X	X
tpf_decb_create	Creates a DECB		X	X
tpf_decb_locate	Locates a DECB		X	X
tpf_decb_release	Releases a DECB		X	X
tpf_decb_swapblk	Swaps a storage level between a DECB and the ECB		X	X
tpf_decb_validate	Validates a DECB		X	X

Figure 4 (Page 9 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
▪ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
tpf_fa4x4c	Converts an 8-byte to a 4-byte file address or a 4-byte to an 8-byte file address		X	X
tpf_fac8c	Computes an online 8-byte file address		X	X
tpf_rcrfc	Releases a pool-file record address and a storage block		X	X
tpf_STCK	Gets S/390 TOD clock value		X	X
tppc_activate_ on_confirmation	Activates a program after confirmation received			X
tppc_activate_ on_receipt	Activates a program after information received			X
tppc_allocate	Allocates a conversation			X
tppc_confirm	Sends confirmation request			X
tppc_confirmed	Sends a confirmation reply			X
tppc_deallocate	Deallocates a conversation			X
tppc_flush	Flushes data from a local LU's buffer			X
tppc_get_attributes	Gets information about a conversation			X
tppc_post_on_receipt	Sets posting active for a conversation			X
tppc_prepare_to_receive	Changes to receive status			X
tppc_test	Tests conversation			X
tppc_receive	Receives information			X
tppc_request_to_send	Requests change to send status			X
tppc_send_data	Sends data to remote application program			X
tppc_send_error	Sends error notification			X
tppc_wait	Waits for posting			X
tprdc	Reads a record from a general sequential file		X	X
trewc	Rewinds a tape			X
trsvc	Reserves a general sequential file		X	X
tsync	Synchronizes a tape			X
twrtc	Writes a record to a general sequential file		X	X
uatbc	Requests MDBF user attribute			X
unfrc	Unholds a file address		X	X
unfrc_ext	Unholds a file address, with extended options			X
ungetc	Pushes a character back onto a specified input	√	M	
unhka	Unhooks storage block			X
va_arg	Allows access to variable number of function arguments	√	√	√

Figure 4 (Page 10 of 10). Comparison of C library functions available under ISO-C, ALCS, and TPF.

Key
√ *Fully compatible with ISO-C standards*
X Supported but not an ISO-C standard
▪ *Equivalent to ISO-C standard but works in a nonstandard way*
M *Fully compatible with ISO-C standards but supported for memory files only.*

FUNCTION NAME	FUNCTION DESCRIPTION	C	ALCS	TPF
va_end	Allows access to variable number of function arguments	√	√	√
va_start	Allows access to variable number of function arguments	√	√	√
vfprintf	Formats and prints characters to the output stream using a variable number of arguments	√	M	
vprintf	Formats and prints characters to stdout using a variable number of arguments	√	√	
vsprintf	Formats and stores characters in a buffer using a variable number of arguments	√	√	√
waitc	Waits for all outstanding I/O requests to complete		X	X
wcscat	Concatenates wchar_t strings	√	√	
wcschr	Searches wchar_t strings for character	√	√	
wcscmp	Compares wchar_t strings	√	√	
wcscpy	Copies wchar_t string	√	√	
wcscspn	Searches wchar_t string for characters	√	√	
wcslen	Reads length of wchar_t string	√	√	
wcsncat	Concatenates wchar_t string segment	√	√	
wcsncmp	Compares wchar_t string segments	√	√	
wcsncpy	Copies wchar_t segments	√	√	
wcspbrk	Locates wchar_t characters in a string	√	√	
wcsrchr	Locates a wchar_t character in a string	√	√	
wcsspn	Finds number of wchar_t characters	√	√	
wcstombs	Converts wchar_t characters to multibyte string	√	√	√ ISO-C
wcswcs	Locates a wchar_t string in another wchar_t string	√	√	
wctomb	Converts wchar_t characters to multibyte characters	√	√	√ ISO-C
wgtac	Locates terminal entry			X
wtopc	Sends a message to a terminal		X	X
wtopc_insert_header	Saves a header for wtopc		X	X
wtopc_routing_list	Saves a routing list for wtopc		X	X
wtopc_text	Sends a message to a terminal		X	X

Appendix B. Acronyms and abbreviations

The following acronyms and abbreviations are used in books of the ALCS Version 2 library. Not all are necessarily present in this book.

AAA	agent assembly area
ACB	VTAM access method control block
ACF	Advanced Communications Function
ACF/NCP	Advanced Communications Function for the Network Control Program, usually referred to simply as “NCP”
ACF/VTAM*	Advanced Communications Function for the Virtual Telecommunication Access Method, usually referred to simply as “VTAM”
ACK	positive acknowledgment (SLC LCB)
ACP	Airline Control Program
AID	IBM 3270 attention identifier
AIX	add item index
ALC	airlines line control
ALCI	Airlines Line Control Interconnection
ALCS/MVS/XA	Airline Control System/MVS/XA
ALCS/VSE	Airline Control System/Virtual Storage Extended
ALCS V2	Airline Control System Version 2
AML	acknowledge message label (SLC LCB)
AMS	access method services
AMSG	AMSG application message format
APAR	authorized program analysis report
APF	authorized program facility
API	application program interface
APPC	advanced program-to-program communications
ARINC**	Aeronautical Radio Incorporated
ASCU	agent set control unit (SITA), a synonym for “terminal control unit”
AT&T**	American Telephone and Telegraph Co.
ATA	Air Transport Association of America
ATSN	acknowledge transmission sequence number (SLC)
BATAP	Type B application-to-application program
BSC	binary synchronous communication
C	C programming language
CAF	DB2 Call Attach Facility
CCW	channel command word
CDPI	clearly differentiated programming interface
CEC	central electronic complex
CEUS	communication end-user system
CI	VSAM control interval
CICS*	Customer Information Control System
CLIST	command list
CMC	communication management configuration
CML	clear message label (synonym for AML)
COBOL	COmmon Business Oriented Language
CPI-C	Common Programming Interface – Communications
CPU	central processing unit
CRAS	computer room agent set
CRI	communication resource identifier

CRN	communication resource name
CSA	common service area
CSECT	control section
CSID	cross system identifier
CSW	channel status word
CTKB	Keypoint record B
CTL	control system error
CUA*	Common User Access
DASD	direct access storage device
DBCS	double-byte character set
DBRM	DB2 database request module
DB2*	IBM DB2 for z/OS
DCB	data set control block
DECB	ALCS data event control block
DF	delayed file record
DFDSS	Data Facility Data Set Services
DFHSM	Data Facility Hierarchical Storage Manager
DFP	Data Facility Product
DFSMS*	Data Facility Storage Management Subsystem
DFT	distributed function terminal
DIX	delete item index
DRIL	data record information library
DSI	direct subsystem interface
DSECT	dummy control section
DTP	ALCS diagnostic file processor
EBCDIC	extended binary-coded decimal interchange code
ECB	ALCS entry control block
EIB	error index byte
EID	event identifier
EJB	Enterprise Java Bean
ENQ	enquiry (SLC LCB)
EOF	end of file
EOM	end of message
EOI	end of message incomplete
EOP	end of message pushbutton
EOU	end of message unsolicited
EP	Emulation Program
EP/VS	Emulation Program/VS
ETX	end of text
EVCB	MVS event control block
EXCP	Execute Channel Program
FACE	file address compute
FIFO	first-in-first-out
FI	file immediate record
FM	function management
FMH	function management header
GB	gigabyte (1 073 741 824 bytes)
GDS	general data set
GFS	get file storage (called pool file storage in ALCS)
GMT	Greenwich Mean Time
GTF	generalized trace facility (MVS)
GUPI	general-use programming interface
HEN	high-level network entry address
HEX	high-level network exit address

HFS	Hierarchical File System
HLASM	High Level Assembler
HLL	high-level language
HLN	high-level network
HLS	high-level system (for example, SITA)
IA	interchange address
IASC	International Air Transport Solution Centre
IATA	International Air Transport Association
IATA5	ATA/IATA transmission code 5
IATA7	ATA/IATA transmission code 7
ICF	integrated catalog facility
ID	identifier
ILB	idle (SLC LCB)
IMA	BATAP acknowledgement
IMS*	Information Management System
IMSG	IMSG input message format
I/O	input/output
IOCB	I/O control block
IP	Internet Protocol
IPARS	International Programmed Airlines Reservation System
IPCS	Interactive Problem Control System
IPL	initial program load
ISA	initial storage allocation
ISC	intersystem communication
ISO/ANSI	International Standards Organization/American National Standards Institute
ISPF	Interactive System Productivity Facility
ISPF/PDF	Interactive System Productivity Facility/Program Development Facility
ITA2	International Telegraph Alphabet number 2
JCL	job control language
JES	job entry subsystem
JNDI	Java Naming and Directory Interface
KB	kilobyte (1024 bytes)
KCN	link channel number (SLC)
KSDS	VSAM key-sequenced data set
LAN	local area network
LCB	link control block (SLC)
LDB	link data block (SLC)
LDI	local DXCREI index
LEID	logical end-point identifier
LE	Language Environment*
LICRA	Link Control – Airline
LMT	long message transmitter
LN	line number (ALCS/VSE and TPF terminology)
LN/ARID	line number and adjusted resource identifier (ALCS/VSE terminology)
LSET	Load set
LSI	link status identifier (SLC)
LU	logical unit
LU 6.2	Logical Unit 6.2
MATIP	Mapping of airline traffic over IP
MB	megabyte (1 048 576 bytes)
MBI	message block indicator (SLC)

MCHR	module/cylinder/head/record
MESW	message switching
MNOTE	message note
MQI	Message Queueing Interface
MQM	Message Queue Manager
MSNF	Multisystem Networking Facility
MVS*	Multiple Virtual Storage (refers to both MVS/XA and MVS/ESA, and also to OS/390* and z/OS*)
MVS/DFP*	Multiple Virtual Storage/Data Facility Product
MVS/ESA*	Multiple Virtual Storage/Enterprise System Architecture
MVS/XA*	Multiple Virtual Storage/Extended Architecture
NAB	next available byte
NAK	negative acknowledgment (SLC LCB)
NCB	network control block (SLC)
NCP	Network Control Program (refers to ACF/NCP)
NCP/VS	Network Control Program/Virtual Storage.
NEF	Network Extension Facility
NEF2	Network Extension Facility 2
NPDA	Network Problem Determination Application
NPSI	Network Control Program packet switching interface
NTO	Network Terminal Option
OCR	one component report
OCTM	online communication table maintenance
OLA	optimized local adapters
OMSG	OMSG output message format
OPR	operational system error
OSID	other-system identification
OS/2*	IBM Operating System/2
PARS	Programmed Airlines Reservation System
PDF	parallel data field (refers to NCP)
PDM	possible duplicate message
PDS	partitioned data set
PDSE	partitioned data set extended
PDU	pool directory update
PER	program event recording
PFDR	pool file directory record
PL/I	programming language one
PLM	purge long message (name of ALCS/VSE and TPF general tape)
PLU	primary logical unit
PNL	passenger name list
PNR	passenger name record
PP	IBM program product
PPI	program-to-program interface
PPMSG	program-to-program message format
PPT	program properties table
PR	permanently resident record
PRC	prime computer room agent set
PRDT	physical record (block) descriptor table
PRPQ	programming request for price quotation
PR/SM*	Processor Resource/Systems Manager*
PS	VTAM presentation services
PSPI	product sensitive programming interface
PSW	program status word
PTF	program temporary fix

PTT	Post Telephone and Telegraph Administration
PU	physical unit
PVC	permanent virtual circuit
QSAM	queued sequential access method
RACF*	resource access control facility
RB	request block
RBA	relative byte address
RCC	record code check
RCPL	routing control parameter list
RCR	resource control record
RCS	regional control center
RDB	Relational Database
RDBM	Relational Database Manager
REI	resource entry index
RLT	record locator table
RMF*	Resource Measurement Facility*
RO CRAS	receive-only computer room agent set
RON	record ordinal number
RPL	VTAM request parameter list
RPQ	request for price quotation
RSM	resume (SLC LCB)
RTM	recovery and termination management
RU	request unit
SAA*	Systems Application Architecture*
SAF	System Authorization Facility
SAL	system allocator list (TPF terminology)
SAM	sequential access method
SDLC	Synchronous Data Link Control
SDMF	standard data and message file
SDSF	System Display and Search Facility
SDWA	system diagnostic work area
SI	DBCS shift in
SITA**	Société Internationale de Télécommunications Aéronautiques
SLC	ATA/IATA synchronous link control
SLIP	serviceability level indication processing
SLN	symbolic line number
SLR	Service Level Reporter
SLU	secondary logical unit
SMP/E	System Modification Program Extended
SNA	Systems Network Architecture
SO	DBCS shift out
SON	system ordinal number
SQA	system queue area
SQL	Structured Query Language
SQLCA	SQL Communication Area
SQLDA	SQL Descriptor Area
SRB	service request block
SRG	statistical report generator
SRM	System Resource Manager
STC	system test compiler
STP	stop (SLC LCB)
STV	system test vehicle
SWB	service work block
SYN	character synchronization character

TA	terminal address
TAS	time available supervisor
TCB	task control block
TCID	terminal circuit identity
TCP/IP	Transmission Control Protocol / Internet Protocol
TI	time-initiated record
TOD	time of day
TPF	Transaction Processing Facility
TPF/APPC	Transaction Processing Facility/Advanced Program to Program Communications
TPF/DBR	Transaction Processing Facility/Data Base Reorganization
TPFDF	TPF Database Facility
TPF/MVS	Transaction Processing Facility/MVS (alternative name for ALCS V2)
TP_ID	transaction program identifier
TSI	transmission status indicator
TSN	transmission sequence number
TSO	time-sharing option
TSO/E	Time Sharing Option Extensions
TUT	test unit tape (sequential file)
UCB	unit control block
UCTF	Universal Communications Test Facility
VFA	virtual file access
VIPA	virtual IP address
VM	virtual machine
VM/CMS	virtual machine/conversational monitor system
VS	virtual storage
VSAM	virtual storage access method
VSE	Virtual Storage Extended
VSE/AF	Virtual Storage Extended/Advanced Function
VSE/VSAM	Virtual Storage Extended/Virtual Storage Access Method
VTAM*	Virtual Telecommunications Access Method (refers to VTAM)
VTOC	volume table of contents
WAS	WebSphere Application Server
WSF	Write Structured Field
WTTY	World Trade Teletypewriter
XMSG	XMSG message switching message format
XREF	ALCS cross referencing facility

Glossary

Notes:

1. Acronyms and abbreviations are listed separately from this Glossary. See Appendix B, “Acronyms and abbreviations” on page 243.
2. For an explanation of any term not defined here, see the IBM *Dictionary of Computing*.

A

AAA hold. See terminal hold.

abnormal end of task (abend). Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

access method services (AMS). A utility program that defines VSAM data sets (or files) and allocates space for them, converts indexed sequential data sets to key-sequenced data sets with indexes, modifies data set attributes in the catalog, facilitates data set portability between operating systems, creates backup copies of data sets and indexes, helps make inaccessible data sets accessible, and lists data set records and catalog entries.

activity control variable. A parameter that ALCS uses to control its workload. The system programmer defines activity control variables in the ALCS system configuration table generation.

Advanced Communications Function for the Network Control Program (ACF/NCP). An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

Advanced Program-to-Program Communications (APPC). A set of inter-program communication services that support cooperative transaction processing in an SNA network. APPC is the implementation, on a given system, of SNA’s logical unit type 6.2 (LU 6.2). See APPC component and APPC transaction scheduler.

Aeronautical Radio Incorporated (ARINC). An organization which provides communication facilities for use within the airline industry.

agent assembly area (AAA). A fixed-file record used by IPARS applications. One AAA record is associated with each terminal and holds data that needs to be kept beyond the life of an entry. For example, to collect information from more than one message.

agent set. Synonym for communication terminal.

agent set control unit (ASCU). Synonym for terminal interchange.

Airline Control Program (ACP). An earlier version of the IBM licensed program Transaction Processing Facility (TPF).

Airline Control System (ALCS). A transaction processing platform providing high performance, capacity, and availability, that runs specialized (typically airline) transaction processing applications.

Airline Control System/Multiple Virtual Storage/Extended Architecture (ALCS/MVS/XA). An ALCS release designed to run under an MVS/XA operating system.

Airline Control System Version 2 (ALCS V2). An ALCS release designed to run under a z/OS operating system.

Airline Control System/Virtual Storage Extended (ALCS/VSE). An ALCS release designed to run under a VSE/AF operating system.

airlines line control (ALC). A communication protocol particularly used by airlines.

Airlines Line Control Interconnection (ALCI). A feature of Network Control Program (NCP) that allows it to manage ALC networks in conjunction with a request for price quotation (RPQ) scanner for the IBM 3745 communication controller.

Airline X.25 (AX.25). A discipline conforming to the ATA/IATA AX.25 specification in the ATA/IATA publication *ATA/IATA Interline Communications Manual*, ATA/IATA document DOC.GEN 1840. AX.25 is based on X.25 and is intended for connecting airline computer systems to SITA or ARINC networks.

ALCS command. A command addressed to the ALCS system. All ALCS commands start with the letter Z (they are also called “Z messages”) and are 5 characters long.

These commands allow the operator to monitor and control ALCS. Many of them can only be entered from CRAS terminals. ALCS commands are called “functional messages” in TPF.

ALCS data collection file. A series of sequential data sets to which ALCS writes performance-related data for subsequent processing by the statistical report

generator or other utility program. See also data collection and statistical report generator.

ALCS diagnostic file. A series of sequential data sets to which the ALCS monitor writes all types of diagnostic data for subsequent processing by the diagnostic file processor.

ALCS diagnostic file processor. An offline utility, often called the “post processor”, that reads the ALCS diagnostic file and formats and prints the dump, trace, and system test vehicle (STV) data that it contains.

ALCS entry dispatcher. The ALCS online monitor's main work scheduler. Often called the “CPU loop”.

ALCS offline program. An ALCS program that runs as a separate MVS job (not under the control of the ALCS online monitor).

ALCS online monitor. The part of ALCS that performs the services for the ECB-controlled programs and controls their actions.

ALCS trace facility. An online facility that monitors the execution of application programs. When it meets a selected monitor-request macro, it interrupts processing and sends selected data to an ALCS display terminal, to the ALCS diagnostic file, or to the system macro trace block. See also instruction step.

The ALCS trace facility also controls tracing to the MVS generalized trace facility (GTF), for selected VTAM communication activity.

ALCS update log file. A series of sequential data sets in which the ALCS monitor records changes to the real-time database.

ALCS user file. A series of sequential data sets to which you may write all types of diagnostic data for subsequent processing by an offline processor. You write the data from an installation-wide monitor exit using the callable service UWSEQ.

allocatable pool. The ALCS record class that includes all records on the real-time database. Within this class, there is one record type for each DASD record size.

The allocatable pool class is special in that ALCS itself can dispense allocatable pool records and use them for other real-time database record classes. For example, all fixed-file records are also allocatable pool records (they have a special status of “in use for fixed file”).

When ALCS is using type 2 long-term pool dispense, ALCS satisfies requests for long-term pool by dispensing available allocatable pool records.

See DASD record, real-time database, record class, and record type.

alternate CRAS. A computer room agent set (CRAS) that is not Prime CRAS or receive only CRAS. See computer room agent set, Prime CRAS, and receive only CRAS.

alternate CRAS printer. A CRAS printer that is not receive only CRAS. See CRAS printer and receive only CRAS.

answerback. A positive acknowledgement (ACK) from an ALC printer.

APPC component. The component of MVS that is responsible for extending LU 6.2 and SAA CPI Communications services to applications running in any MVS address space. Includes APPC conversations and scheduling services.

APPC transaction scheduler. A program such as ALCS that is responsible for scheduling incoming work requests from cooperative transaction programs.

application plan. See DB2 application plan.

application. A group of associated application programs that carry out a specific function.

application global area. An area of storage in the ALCS address space containing application data that any entry can access.

The application global area is subdivided into keypointable and nonkeypointable records. Keypointable records are written to the database after an update; nonkeypointable records either never change, or are reinitialized when ALCS restarts.

C programs refer to global records and global fields within the application global area.

application program. A program that runs under the control of ALCS. See also ECB-controlled program.

application program load module. In ALCS, a load module that contains one or more application programs.

application queue. In message queuing with ALCS, any queue on which application programs put and get messages using MQI calls.

assign. Allocate a general sequential file to an entry. The `TOPNC` monitor-request macro (or equivalent C function) opens and allocates a general sequential file. The `TASNC` monitor-request macro (or equivalent C function) allocates a general sequential file that is already open but not assigned to an entry (it is reserved).

associated resource. Some ALCS commands generate output to a printer (for example, `ZDCOM` prints information about a communication resource). For this type of command the printed output goes to the

associated resource; that is, to a printer associated with the originating display. There is also a response to the originating display that includes information identifying the associated resource.

asynchronous trace. One mode of operation of the ALCS trace facility. Asynchronous trace is a conversational trace facility to interactively trace entries that do not originate from a specific terminal.

automatic storage block. A storage block that is attached to an entry, but is not attached at a storage level. An assembler program can use the ALASC monitor-request macro to obtain an automatic storage block and BACKC monitor-request macro to release it. C programs cannot obtain automatic storage blocks.

B

backward chain. The fourth fullword of a record stored on the ALCS database, part of the record header. See chaining of records.

When standard backward chaining is used, this field contains the file address of the previous record in the chain, except that the first record contains the file address of the last record in the chain. (If there is only one record, the backward chain field contains zeros.)

balanced path. A path where no single component (channel, DASD director or control unit, head of string, and internal path to the DASD device) is utilized beyond the limits appropriate to the required performance.

bar. In the MVS 64-bit address space, a virtual line called the bar marks the 2-gigabyte address. The bar separates storage below the 2-gigabyte address, called **below the bar**, from storage above the 2-gigabyte address, called **above the bar**.

BATAP. Type B application-to-application program

Binary Synchronous Communication (BSC). A form of telecommunication line control that uses a standard set of transmission control characters and control character sequences, for binary synchronous transmission of binary-coded data between stations.

bind. See DB2 bind

BIND. In SNA, a request to activate a session between two logical units (LUs). The BIND request is sent from a primary LU to a secondary LU. The secondary LU uses the BIND parameters to help determine whether it will respond positively or negatively to the BIND request.

binder. The program that replaces the linkage editor and batch loader programs that were provided with earlier versions of MVS.

BIND image. In SNA, the set of fields in a BIND request that contain the session parameters.

block. See storage block.

C

catastrophic. A type of system error that results in the termination of ALCS.

chain-chase. See Recoup.

chaining of records. One record can contain the file address of another (usually a pool-file record). The addressed record is said to be chained from the previous record. Chains of records can contain many pool-file records. See forward chain and backward chain.

class. See record class.

clearly differentiated programming interfaces (CDPI). A set of guidelines for developing and documenting product interfaces so that there is clear differentiation between interfaces intended for general programming use (GUPIs) and those intended for other specialized tasks.

close. Close a sequential file data set (MVS CLOSE macro) and deallocate it from ALCS. For general sequential files this is a function of the TCLSC monitor-request macro (or equivalent C function). ALCS automatically closes other sequential files at end-of-job.

command. See ALCS command.

command list (CLIST). A sequential list of commands, control statements, or both, that is assigned a name. When the name is invoked the commands in the list are executed.

commit. An operation that terminates a unit of recovery. Data that was changed is now consistent.

common entry point (CEP). A function in the Transaction Processing Facility Database Facility (TPFDF) product that provides common processing for all TPFDF macro calls issued by ALCS application programs. It also provides trace facilities for TPFDF macro calls.

Common Programming Interface – Communications (CPI-C). The communication element of IBM Systems Application Architecture (SAA). CPI-C provides a programming interface that allows program-to-program communication using the IBM SNA logical unit 6.2.

Common User Access. Guidelines for the dialog between a user and a workstation or terminal.

communication management configuration (CMC). A technique for configuring a network that allows for the consolidation of many network management functions for the entire network in a single host processor.

communication resource. A communication network component that has been defined to ALCS. These include each terminal on the network and other network components that ALCS controls directly (for example, SLC links). Resources can include, for example:

SNA LUs (including LU 6.1 links)
ALC terminals
SLC and WTTY links
Applications.

communication resource identifier (CRI). A 3-byte field that uniquely identifies an ALCS communication resource. It is equivalent to the LN/IA/TA in TPF and the LN/ARID in ALCS/VSE. ALCS generates a CRI for each resource.

communication resource name (CRN). A 1- to 8-character name that uniquely identifies an ALCS communication resource. For SNA LUs, it is the LU name. The system programmer defines the CRN for each resource in the ALCS communication generation.

communication resource ordinal. A unique number that ALCS associates with each communication resource. An installation can use the communication resource ordinal as a record ordinal for a particular fixed-file record type. This uniquely associates each communication resource with a single record.

For example, IPARS defines a fixed-file record type (#WAARI) for AAA records. Each communication resource has its own AAA record — the #WAARI record ordinal is the communication resource ordinal. See also record ordinal and agent assembly area.

compiler. A program that translates instructions written in a high level programming language into machine language.

computer room agent set (CRAS). An ALCS terminal that is authorized for the entry of restricted ALCS commands.

Prime CRAS is the primary terminal that controls the ALCS system. Receive Only CRAS (RO CRAS) is a designated printer or NetView operator identifier to which certain messages about system function and progress are sent.

configuration data set. (1) A data set that contains configuration data for ALCS. See also configuration-dependent table. (2) The ALCS record class that includes all records on the configuration data set. There is only one record type for this class. See record class and record type.

configuration-dependent table. A table, constructed by the ALCS generation process, which contains configuration-dependent data. Configuration-dependent tables are constructed as conventional MVS load modules. In ALCS V2, there are separate configuration-dependent tables for:

System data
DASD data
Sequential file data
Communication data
Application program data.

See also configuration data set.

control byte. The fourth byte of a record stored on the ALCS database, part of the record header. ALCS ignores this byte; some applications, however, make use of it.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records. The control interval is the unit of information that VSAM transmits to or from direct access storage.

control transfer. The process that the ALCS online monitor uses to create a new entry and to transfer control to an ECB-controlled program.

conversation_ID:. An 8-byte identifier, used in Get_Conversation calls, that uniquely identifies a conversation. APPC/MVS returns a conversation_ID on the CMINIT, ATBALLOC, and ATBGETC calls; a conversation_ID is required as input on subsequent APPC/MVS calls.

CPU loop. See ALCS entry dispatcher.

CRAS printer. A computer room agent set (CRAS) that is a printer terminal. See computer room agent set.

CRAS display. A computer room agent set (CRAS) that is a display terminal. See computer room agent set.

CRAS fallback. The automatic process that occurs when the Prime CRAS or receive only CRAS becomes unusable by which an alternate CRAS becomes Prime CRAS or receive only CRAS. See also Prime CRAS, receive only CRAS, and alternate CRAS.

create service. An ALCS service that enables an ALCS application program to create new entries for asynchronous processing. The new ECBs compete for system resources and, once created, are not dependent or connected in any way with the creating ECB.

cycling the system. The ALCS system can be run in one of four different system states. Altering the system state is called cycling the system. See SLC link for another use of the term “cycling”.

D

DASD record. A record stored on a direct access storage device (DASD). ALCS allows the same range of sizes for DASD records as it allows for storage blocks, except no size L0 DASD records exist.

data collection. An online function that collects data about selected activity in the system and sends it to the ALCS data collection file, if there is one, or to the ALCS diagnostic file. See also statistical report generator.

database request module (DBRM). A data set member created by the DB2 precompiler that contains information about SQL statements. DBRMs are used in the DB2 bind process. See DB2 bind.

data-collection area. An ECB area used by the ALCS online monitor for accumulating statistics about an entry.

data event control block (DECB). An ALCS control block, that may be acquired dynamically by an entry to provide a storage level and data level in addition to the 16 ECB levels. It is part of entry storage.

The ALCS DECB is independent of the MVS control block with the same name.

Data Facility Storage Management Subsystem (DFSMS*). An MVS operating environment that helps automate and centralize the management of storage. It provides the storage administrator with control over data class, management class, storage group, and automatic class selection routine definitions.

Data Facility Sort (DFSORT*). An MVS utility that manages sorting and merging of data.

data file. A sequential data set, created by the system test compiler (STC) or by the `ZDATA DUMP` command, that contains data to be loaded on to the real-time database. (An ALCS command `ZDATA LOAD` can be used to load data from a data file to the real-time database.) A data file created by STC is also called a “pilot” or “pilot tape”.

data level. An area in the ECB or a DECB used to hold the file address, and other information about a record. See ECB level and DECB level.

data record information library (DRIL). A data set used by the system test compiler (STC) to record the formats of data records on the real-time system. DRIL is used when creating data files.

DB2 application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered during program execution. See DB2 bind.

DB2 bind. The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

DB2 Call Attach Facility (CAF). An interface between DB2 and batch address spaces. CAF allows ALCS to access DB2.

DB2 for z/OS. An IBM licensed program that provides relational database services.

DB2 host variable. In an application program, an application variable referenced by embedded SQL statements.

DB2 package. Also called application package. An object containing a set of SQL statements that have been bound statically and that are available for processing. See DB2 bind.

DB2 package list. An ordered list of package names that may be used to extend an application plan.

DECB level. When an application program, running under ALCS, reads a record from a file, it must “own” a storage block in which to put the record. The address of the storage block may be held in an area of a DECB called a storage level.

Similarly, there is an area in a DECB used for holding the 8-byte file address, record ID, and record code check (RCC) of a record being used by an entry. This is a data level.

The storage level and data level in a DECB, used together, are called a DECB level.

See also ECB level.

diagnostic file. See ALCS diagnostic file.

dispatching priority. A number assigned to tasks, used to determine the order in which they use the processing unit in a multitasking situation.

dispense (a pool-file record). To allocate a long-term or short-term pool-file record to a particular entry. ALCS performs this action when requested by an application program. See release a pool-file record.

double-byte character set. A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

duplex. A communication link on which data can be sent and received at the same time. Synonymous with full duplex. Communication in only one direction at a time is called “half-duplex”. Contrast with simplex transmission.

duplex database. Synonym for duplicated database.

duplicated database. A database where each data set is a mirrored pair. In ALCS, you can achieve this using either ALCS facilities or DASD controller facilities (such as the IBM 3990 dual copy facility). See mirrored pair.

dynamic program linkage. Program linkage where the connection between the calling and called program is established during the execution of the calling program. In ALCS dynamic program linkage, the connection is established by the ALCS ENTER/BACK services. Contrast with static program linkage.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution. Contrast with embedded SQL.

E

ECB-controlled program. A program that runs under the control of an entry control block (ECB). These programs can be application programs or programs that are part of ALCS, for example the ALCS programs that process operator commands (Z messages). ECB-controlled programs are known as E-type programs in TPF.

ECB level. When an application program, running under ALCS, reads a record from file, it must “own” a storage block in which to put the record. The address of the storage block may be held in an area of the ECB called a storage level.

There are 16 storage levels in the ECB. A storage block with its address in slot zero in the ECB is said to be attached on level zero.

Similarly, there are 16 areas in the ECB that may be used for holding the 4-byte file addresses, record ID, and record code check (RCC) of records being used by an entry. These are the 16 data levels.

Storage levels and data levels, used together, are called ECB levels.

See also DECB level.

embedded SQL. Also called static SQL. SQL statements that are embedded within an application program and are prepared during the program preparation process before the program is executed. After it is prepared, the statement itself does not change (although values of host variables specified within the statement can change). Contrast with dynamic SQL.

Emulation Program/Virtual Storage (EP/VS). A component of NCP/VS that ALCS V2 uses to access SLC networks.

ENTER/BACK. The general term for the application program linkage mechanism provided by ALCS.

entry. The basic work scheduling unit of ALCS. An entry is represented by its associated entry control block (ECB). It exists either until a program that is processing that entry issues an `EXITC` monitor-request macro (or equivalent C function), or until it is purged from the system. An entry is created for each input message, as well as for certain purposes unrelated to transactions. One transaction can therefore generate several entries.

entry control block (ECB). A control block that represents a single entry during its life in the system.

entry dispatcher. See ALCS entry dispatcher.

entry macro trace block. There is a macro trace block for each entry. Each time an entry executes a monitor-request macro (or a corresponding C function), ALCS records information in the macro trace block for the entry.

This information includes the macro request code, the name of the program that issued the macro, and the displacement in the program. The ALCS diagnostic file processor formats and prints these macro trace blocks in ALCS system error dumps.

See also system macro trace block.

entry storage. The storage associated with an entry. It includes the ECB for the entry, storage blocks that are attached to the ECB or DECBs, storage blocks that are detached from the ECB or DECBs, automatic storage blocks, and DECBs. It also includes heap storage (for high-level language or assembler language programs) and stack storage (for high-level language programs).

equate. Informal term for an assignment instruction in assembler languages.

error index byte (EIB). See SLC error index byte.

extended buffer. A storage area above 2 GB used for large messages.

extended message format. For input and output messages, a message format which includes a 4-byte field for the message length.

Execute Channel Program (EXCP). An MVS macro used by ALCS V2 to interface to I/O subsystems for SLC support.

F

fetch access. Access which only involves reading (not writing). Compare with store access.

file address. 4-byte (8 hexadecimal digits) value or 8-byte value in 4x4 format (low order 4-bytes contain a 4-byte file address, high order 4 bytes contain hexadecimal zeros) that uniquely identifies an ALCS record on DASD. `FIND/FILE` services use the file address when reading or writing DASD records. See fixed file and pool file.

file address compute routine (FACE). An ALCS routine, called by a monitor-request macro (or equivalent C function) that calculates the file address of a fixed-file record. The application program provides the FACE routine with the fixed-file record type and the record ordinal number. FACE returns the 4-byte file address.

There is also an `FAC8C` monitor-request macro (or equivalent C function), that will return an 8-byte file address in 4x4 format.

FIND/FILE. The general term for the DASD I/O services that ALCS provides.

fixed file. An ALCS record class – one of the classes that reside on the real-time database. All fixed-file records are also allocatable pool records (they have a special status of “in use for fixed file”).

Within this class there are two record types reserved for use by ALCS itself (#KPTRI and #CPRCR). There can also be installation-defined fixed-file record types.

Each fixed-file record type is analogous to a relative file. Applications access fixed-file records by specifying the fixed-file record type and the record ordinal number. Note however that fixed-file records are not physically organized as relative files (logically adjacent records are not necessarily physically adjacent).

See real-time database, record class, and record type. See also system fixed file. Contrast with pool file.

fixed-file record. One of the two major types of record in the real-time database (the other is a pool-file record). When the number of records of a particular kind will not vary, the system programmer can define a fixed file record type for these records. ALCS application programs accessing fixed-file records use the `ENTRC` monitor-request macro to invoke the 4-byte file address compute routine (FACE or FACS) or use the `FAC8C` monitor-request macro to compute an 8-byte file address. The equivalent C functions are `face` or `facs` or `tpf_fac8c`.

fixed-file record type. (Known in TPF as FACE ID.) The symbol, by convention starting with a hash sign (#)[1] which identifies a particular group of fixed-file records. It is called the fixed-file record type symbol. The equated value of this symbol (called the fixed-file record type value) also identifies the fixed-file record type.

forward chain. The third fullword of a record stored on the ALCS database (part of the record header). When standard forward chaining is used, this field contains the file address of the next record in the chain, except that the last (or only) record contains binary zeros.

full-duplex. Deprecated term for duplex.

functional message. See ALCS command.

G

general data set (GDS). The same as a general file, but accessed by different macros or C functions in ALCS programs.

general file. (1) A DASD data set (VSAM cluster) that is used to communicate data between offline utility programs and the online system. General files are not part of the real-time database. (2) The ALCS record class that includes all records on the general files and general data sets. Each general file and general data set is a separate record type within this class. See record class and record type.

general file record. A record on a general file.

generalized trace facility (GTF). An MVS trace facility. See also ALCS trace facility.

general sequential file. A class of sequential data set that is for input or output. ALCS application programs must have exclusive access to a general sequential file before they can read or write to it. See also real-time sequential file.

general tape. TPF term for a general sequential file.

general-use programming interface (GUPI). An interface intended for general use in customer-written applications.

get file storage (GFS). The general term for the pool file dispense mechanisms that ALCS provides.

---

[1] This character might appear differently on your equipment. It is the character represented by hexadecimal 7B.

global area. See application global area.

global resource serialization. The process of controlling access of entries to a global resource so as to protect the integrity of the resource.

H

half-duplex. A communication link that allows transmission in one direction at a time. Contrast with duplex.

halt. (1) The ALCS state when it is terminated. (2) The action of terminating ALCS.

heap. An area of storage that a compiler uses to satisfy requests for storage from a high-level language (for example, `calloc` or `malloc` C functions). ALCS provides separate heaps for each entry (if needed). The heap is part of entry storage. Assembler language programs may also obtain or release heap storage using the `CALOC`, `MALOC`, `RALOC`, and `FREEC` monitor-request macros.

High Level Assembler (HLASM). A functional replacement for Assembler H Version 2. HLASM contains new facilities for improving programmer productivity and simplifying assembler language program development and maintenance.

high-level language (HLL). A programming language such as C or COBOL.

high-level language (HLL) storage unit. Alternative name for a type 2 storage unit. See storage unit.

high-level network (HLN). A network that provides transmission services between transaction processing systems (for example, ALCS) and terminals. Strictly, the term “high-level network” applies to a network that connects to transaction processing systems using SLC. But in ALCS publications, this term is also used for a network that connects by using AX.25 or MATIP.

high-level network designator (HLD). The entry or exit point of a block in a high-level network. For SLC networks, it is the SLC address of a switching center that is part of a high-level network. It comprises two bytes in the 7-bit transmission code used by SLC.

HLN entry address (HEN). The high-level designator of the switching center where a block enters a high-level network.

HLN exit address (HEX). The high-level designator of the switching center where a block leaves a high-level network.

hold. A facility that allows multiple entries to share data, and to serialize access to the data. The data can be a database record, or any named data resource. This facility can be used to serialize conflicting processes. See also record hold and resource hold.

host variable. See DB2 host variable

I

information block. See SLC link data block.

initial storage allocation (ISA). An area of storage acquired at initial entry to a high-level language program. ALCS provides a separate ISA for each entry (if required). The ISA is part of entry storage.

initiation queue. In message queuing, a local queue on which the queue manager puts trigger messages. You can define an initiation queue to ALCS, in order to start an ALCS application automatically when a trigger message is put on the queue. See trigger message.

input/output control block (IOCB). A control block that represents an ALCS internal “task”. For example, ALCS uses an IOCB to process a DASD I/O request.

input queue. In message queuing with ALCS, you can define a local queue to ALCS in order to start an ALCS application automatically when a message is put on that queue. ALCS expects messages on the input queue to be in PPMSG message format. See PPMSG.

installation-wide exit. The means specifically described in an IBM software product’s documentation by which an IBM software product may be modified by a customer’s system programmers to change or extend the functions of the IBM software product. Such modifications consist of exit routines written to replace an existing module of an IBM software product, or to add one or more modules or subroutines to an IBM software product for the purpose of modifying (including extending) the functions of the IBM software product. Contrast with user exit.

instruction step. One mode of operation of the ALCS trace facility. Instruction step is a conversational trace facility that stops the traced application program before the execution of each processor instruction.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. ISPF provides a means of generating standard screen panels and interactive dialog between the application programmer and terminal user.

interchange address (IA). In ALC, the 1-byte address of a terminal interchange. Different terminal interchanges connected to the same ALC link have different interchange addresses. Different terminal interchanges connected to different ALC links can have

the same interchange address. See also terminal interchange

International Programmed Airlines Reservation System (IPARS). A set of applications for airline use. The principal functions are reservations and message switching.

IPARS for ALCS. The ALCS shipment includes IPARS as a sample application, and installation verification aid for ALCS.

K

KCN. Abbreviation for an SLC channel number. See SLC channel.

keypointable. See application global area.

keypoint B (CTKB). A record that contains dynamic system information that ALCS writes to DASD when it is updated so that ALCS can restart from its latest status.

L

Language Environment*. A common run-time environment and common run-time services for z/OS high level language compilers.

level. See ECB level.

line number (LN). (1) In ALC, the 1-byte address of an ALC link. Different links connected to the same communication controller have different line numbers. Different links connected to different communication controllers can have the same line number.
(2) Synonym for symbolic line number.

Link Control — Airline (LICRA). The name of a programming request for price quotation (PRPQ) to the IBM 3705 Emulation Program (EP/VS). This modifies EP/VS to support SLC networks.

link control block (LCB). See SLC link control block.

link data block (LDB). See SLC link data block.

link trace. See SLC link trace.

local DXCREI index (LDI). The first byte of a communication resource indicator (CRI).

local queue. In message queuing, a queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with remote queue.

lock. A serialization mechanism whereby a resource is restricted for use by the holder of the lock. See also hold.

log. See ALCS update log.

logging. The process of writing copies of altered database records to a sequential file. This is the method used to provide an up-to-date copy of the database should the system fail and the database have to be restored. The database records are logged to the ALCS update log file.

logical end-point identifier (LEID). In NEF2 and ALCI environments, a 3-byte identifier assigned to an ALC terminal.

logical unit type 6.2 (LU 6.2). The SNA logical unit type that supports general communication between programs in a distributed processing environment; the SNA logical unit type on which Common Programming Interface – Communications (CPI-C) is built.

log in. TPF term for establishing routing between a terminal and an application.

log on. Establish a session between an SNA terminal and an application such as ALCS. See also routing.

logon mode. In VTAM, a set of predefined session parameters that can be sent in a BIND request. When a set is defined, a logon mode name is associated with the set.

logon mode table. In VTAM, a table containing several predefined session parameter sets, each with its own logon mode name.

long message transmitter (LMT). A part of the IPARS application that is responsible for blocking and queuing printer messages for output. Also called XLMT.

long-term pool. An ALCS record class – one of the classes that reside on the real-time database. Within this class, there is one record type for each DASD record size. All long-term pool-file records are also allocatable pool records. ALCS application programs can use long-term pool records for long-lived or high-integrity data. See pool file, real-time database, record class, and record type.

L0, L1, L2, L3, ..., L8. Assembler symbols (and defined values in C) for the storage block sizes and record sizes that ALCS supports. See DASD record and storage block size.

M

macro trace block. See entry macro trace block and system macro trace block.

Mapping of Airline Traffic over IP (MATIP). A protocol for transporting traditional airline messages over an IP (Internet Protocol) network. Internet RFC (Request for Comments) number 2351 describes the MATIP protocol.

MBI exhaustion. The condition of an SLC link when a sender cannot transmit another message because all 7 SLC message labels are already "in use"; that is, the sender must wait for acknowledgement of a message so that it can reuse the corresponding message label. See also SLC link, SLC message label, and SLC message block indicator.

message. For terminals with an Enter key, an input message is the data that is sent to the host when the Enter key is hit. A response message is the data that is returned to the terminal. WTTY messages have special "start/end of message" character sequences. One or more input and output message pairs make up a transaction.

message block indicator. See SLC message block indicator.

message label. See SLC message label.

Message Queue Interface (MQI). The programming interface provided by the IBM WebSphere MQ message queue managers. This programming interface allows application programs to access message queuing services.

message queue manager. See queue manager.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues. This enables asynchronous communication between processes that may not be simultaneously active, or for which no data link is active. The message queuing service can assure subsequent delivery to the target application.

message switching. An application that routes messages by receiving, storing, and forwarding complete messages. IPARS for ALCS includes a message switching application for messages that conform to ATA/IATA industry standards for interline communication *ATA/IATA Interline Communications Manual,* DOC.GEN/1840.

mirrored pair. Two units that contain the same data and are referred to by the system as one entity.

monitor-request macro. Assembler language macro provided with ALCS, corresponding to TPF "SVC-type" or "control program" macros. Application programs use these macros to request services from the online monitor.

MQ Bridge. The ALCS MQ Bridge allows application programs to send and receive messages using WebSphere MQ for z/OS queues, without the need to code MQ calls in those programs. The MQ Bridge installation-wide monitor exits USRMQB0, USRMQB1, USRMQB2, and USRMQB3 allow you to customize the behaviour of the MQ Bridge to suit your applications.

MQSeries*. A previous name for WebSphere MQ.

multibyte character. A mixture of single-byte characters from a single-byte character set and double-byte characters from a double-byte character set.

multiblock message. In SLC, a message that is transmitted in more than one link data block. See link data block.

Multiple Virtual Storage/Data Facility Product (MVS/DFP*). An MVS licensed program that isolates applications from storage devices, storage management, and storage device hierarchy management.

Multisystem Networking Facility (MSNF). An optional feature of VTAM that permits these access methods, together with NCP, to control a multiple-domain network.

N

namelist. In message queuing, a namelist is an object that contains a list of other objects.

native file address. For migration purposes ALCS allows two or more file addresses to refer to the same database or general file record. The file address that ALCS uses internally is called the native file address.

NCP Packet Switching Interface (NPSI). An IBM licensed program that allows communication with X.25 lines.

NetView*. A family of IBM licensed programs for the control of communication networks.

NetView operator identifier (NetView operator ID). A 1- to 8-character name that identifies a NetView operator.

NetView program. An IBM licensed program used to monitor a network, manage it, and diagnose network problems.

NetView resource. A NetView operator ID which identifies one of the following:

- A NetView operator logged on to a terminal.
- A NetView operator ID automation task. One of these tasks is used by ALCS to route RO CRAS messages to the NetView Status Monitor Log (STATMON).

network control block (NCB). A special type of message, used for communication between a transaction processing system and a high-level network (HLN). For example, an HLN can use an NCB to transmit information about the network to a transaction processing system.

For a network that connects using SLC, an NCB is an SLC link data block (LDB). Indicators in the LDB differentiate NCBs from other messages.

For a network that connects using AX.25, NCBs are transmitted across a dedicated permanent virtual circuit (PVC).

Network Control Program (NCP). An IBM licensed program resident in an IBM 37xx Communication Controller that controls attached lines and terminals, performs error recovery, and routes data through the network.

Network Control Program Packet Switching Interface (NPSI). An IBM licensed program that provides a bridge between X.25 and SNA.

Network Control Program/Virtual Storage (NCP/VS). An IBM licensed program. ALCS V2 uses the EP/VS component of NCP/VS to access SLC networks.

Network Extension Facility (NEF). The name of a programming request for price quotation (PRPQ P09021) that allows management of ALC networks by NCP; now largely superseded by ALCI.

Network Terminal Option (NTO). An IBM licensed program that converts start-stop terminal device communication protocols and commands into SNA and VTAM communication protocols and commands. ALCS uses NTO to support World Trade Teletypewriter (WTTY).

O

object. In message queuing, objects define the attributes of queue managers, queues, process definitions, and namelists.

offline. A function or process that runs independently of the ALCS online monitor. For example, the ALCS diagnostic file processor is an offline function. See also ALCS offline program.

online. A function or process that is part of the ALCS online monitor, or runs under its control. For example, all ALCS commands are online functions. See also ALCS online monitor.

open. Allocate a sequential file data set to ALCS and open it (MVS `OPEN` macro). For general sequential files this is a function of the `TOPNC` monitor-request macro (or equivalent C function). ALCS automatically opens other sequential files during restart.

optimized local adapters (OLA) for WebSphere Application Server for z/OS (WAS). Built-in, high-speed, bi-directional adapters for calls between WebSphere Application Server for z/OS and ALCS in another address space on the same z/OS image. OLA allows ALCS customers to support an efficient integration of newer Java-based applications with ALCS-based applications. A set of callable services can be used by ALCS assembler or C/C++ programs for exchanging data with applications running in WebSphere Application Server for z/OS. For more information on the callable services (with names of the form BBOA1*xxx*) see the IBM Information Center for WebSphere Application Server - Network Deployment (z/OS) and search for BBOA1. You can use the USRWAS1 installation-wide monitor to verify the caller's authority and to identify input and output messages.

operator command. See ALCS command. Can also refer to non-ALCS commands, for example, MVS or VTAM commands.

ordinal. See communication resource ordinal and record ordinal.

P

package. See DB2 package

package list. See DB2 package list

padded ALC. A transmission code that adds one or more bits to the 6-bit airline line control (ALC) transmission code so that each ALC character occupies one character position in a protocol that uses 7- or 8-bit transmission codes. See also airlines line control.

padded SABRE. Synonym for padded ALC.

passenger name record (PNR). A type of record commonly used in reservation systems. It contains all the recorded information about an individual passenger.

path. The set of components providing a connection between a processor complex and an I/O device. For example, the path for an IBM 3390 DASD volume might include the channel, ESCON Director, 3990 Storage Path, 3390 Device Adapter, and 3390 internal connection. The specific components used in a

particular path are dynamic and may change from one I/O request to the next. See balanced path.

pathlength. The number of machine instructions needed to process a message from the time it is received until the reponse is sent to the communication facilities.

performance monitor. An online function that collects performance data and stores it in records on the ALCS real-time database. It can produce online performance reports based on current data and historical data.

pilot. See data file.

pool directory update (PDU). A facility of TPF that recovers long-term pool file addresses without running Recoup. PDU identifies and makes available all long-term pool-file records that have been released.

pool file. Short-term pool, long-term pool, and allocatable pool. Within each pool file class, there is one record type for each record size; for example, short-term pool includes the record type L1STPOOL (size L1 short-term pool records).

Each pool-file record type contains some records that are in-use and some that are available. There is a dispense function that selects an available record, changes its status to in-use, and returns the file address. Also, there is a release function that takes the file address of an in-use pool-file record and changes the record status to available.

To use a pool-file record, a program must:

1. Request the dispense function. This returns the file address of a record. Note that the record contents are, at this stage, unpredictable.
2. Write the initial record contents, using the file address returned by step 1.
3. Save the file address returned by step 1.
4. Read and write the record to access and update the information as required. These reads and writes use the file address saved in step 3.

When the information in the record is no longer required, a program must:

5. Delete (clear to zeros) the saved copy of the file address (see step 3).
6. Request the release function.

See also record class. Contrast with fixed file.

pool file directory record (PFDR). The ALCS pool file management routine keeps a directory for each size (L1, L2, ...L8) of short-term pool file records and long-term pool-file records. It keeps these directories in pool file directory records.

pool-file record. ALCS application programs access pool-file records with file addresses similar to those for fixed-file records. To obtain a pool-file record, an application program uses a monitor-request macro (or equivalent C function) that specifies a 2-byte record ID or a pool-file record type.

When the data in a pool-file record is no longer required, the application uses a monitor-request macro (or equivalent C function) to release the record for reuse. See pool file.

pool-file record identifier (record ID). The record ID of a pool-file record. On get file requests (using the `GETFC` monitor-request macro or equivalent C function) the program specifies the pool-file record ID. This identifies whether the pool-file record is a short-term or long-term pool-file record and also determines the record size (L1, L2, ...L8). (Coding the 2-byte record IDs, and the corresponding pool-file record sizes and types, is part of the ALCS generation procedure.) See also record ID qualifier.

pool-file record type. Each collection of short-term and long-term pool-file records of a particular record size (identified by the symbols L1, L2, ..., L8) is a different record type. Each pool-file record type has a different name. For short-term pool-file records, this is L*n*STPOOL, where L*n* is the record size symbol. For long-term pool-file records the name is L*n*LTPOOL.

post processor. See ALCS diagnostic file processor.

PPMSG. ALCS program-to-program message format, used by the ALCS message router to send and receive messages on a message routing path to another system. In PPMSG message format, the routing control parameter list (RCPL) precedes the message text.

primary action code. The first character of any input message. The primary action code Z is reserved for ALCS commands. See secondary action code.

Prime CRAS. The primary display terminal, or NetView ID, that controls the ALCS system. See also computer room agent set (CRAS).

process definition object. In message queuing, an object that contains the definition of a message queuing application. For example, a queue manager uses the definition when it works with trigger messages.

product sensitive programming interface (PSPI). An interface intended for use in customer-written programs for specialized purpose only, such as diagnosing, modifying, monitoring, repairing, tailoring or tuning of ALCS. Programs using this interface may need to be changed in order to run with new product releases or versions, or as a result of service.

program linkage. Mechanism for passing control between separate portions of the application program. See dynamic program linkage and static program linkage.

program nesting level. One of 32 ECB areas used by the ENTER/BACK mechanism for saving return control data.

program-to-program interface. In NetView, a facility that allows user programs to send data to, or receive data from, other user programs. It also allows system and application programs to send alerts to the NetView hardware monitor.

P.1024. A SITA implementation of SLC. See SLC.

P.1124. A SITA implementation of SLC. See SLC.

P.1024A. The SITA implementation of airline line control (ALC).

Q

queue manager. A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. WebSphere MQ for z/OS is an example of a queue manager.

R

real-time database. The database to which ALCS must have permanent read and write access. As an ALCS generation option, the real-time database can be duplicated in order to minimize the effects of a DASD failure.

real-time sequential file. A sequential data set used only for output. ALCS application programs can write to any real-time sequential file without requiring exclusive access to the data set. See also general sequential file.

real-time tape. TPF term for a real-time sequential file.

receive only (RO). The function of a communication terminal that can receive but not send data. An example is a printer that does not have a keyboard.

receive only CRAS. A printer terminal (or NetView operator ID) that ALCS uses to direct status messages. Commonly known as RO CRAS.

record. A set of data treated as a unit.

record class. The first (highest) level categorization of ALCS DASD records. ALCS defines the following record classes:

Allocatable pool
Application fixed file
Configuration data set
General file
Long-term pool
Short-term pool
System fixed file.

See also record type and record ordinal.

record code check (RCC). The third byte of any record stored in the ALCS database. It is part of the record header.

The RCC field is intended to help detect the incorrect chaining of records which have the same record ID. This is particularly useful for passenger name records (PNRs), of which there are often hundreds of thousands. A mismatch in RCC values shows that the chain is broken, probably as a result of an application program releasing a record too soon. (A false match cannot be excluded, but the RCC should give early warning of a chaining problem.)

record header. A standard format for the first 16 bytes of a record stored on the ALCS database. It contains the following fields:

Record ID
Record code check
Control byte
Application program name
Forward chain
Backward chain.

Not all records contain forward chains and backward chains. Some applications extend the record header by including extra fields. TPFDF uses an extended record header.

record hold. A type of hold that applies to DASD records. Applications that update records can use record hold to prevent simultaneous updates. See also resource hold.

record identifier (record ID). The first two bytes of a record stored on the ALCS database, part of the record header.

The record ID should always be used to indicate the nature of the data in the record. For example, airlines reservations applications conventionally store passenger name records (PNRs) as long-term pool-file records with a record ID of `'PR'`.

When application programs read such records, they can (optionally) request ALCS to check that the record ID matches that which the application program expects.

When application programs request ALCS to dispense pool file records, ALCS uses the record ID to select an appropriate long-term or short-term pool-file record of the requested record size (L1, L2,...,L8). See also record ID qualifier.

record ID qualifier. A number 0 through 9 that differentiates between record types that have the same record ID.

For compatibility with previous implementations of the record ID qualifier, ALCS also accepts the character qualifiers P and O. P (primary) is equivalent to 0, and O (overflow) is equivalent to 1.

record ordinal. The relative record number within a record type. See record class and record type.

record size. See DASD record.

record type. The second level categorization of ALCS DASD records. Within any one record class, the records are categorized into one or more record types. See also record type number, record type symbol, record class and record ordinal.

record type number. A number that identifies a record type.

record type symbol. The character string that identifies a fixed-file record type (#*xxxxx*), a long-term pool-file record type (LsLTPOOL), a short-term pool-file record type (LsSTPOOL), or a general file (GF-*nnn*). The value of the record type symbol is the record type number.

Recoup. A real-time database validation routine which runs online in the ALCS system. (Note that, while the Recoup routines of TPF consist of a number of phases, some online and some offline, the ALCS Recoup is a single online phase that runs, without operator intervention, in any system state.)

Recoup reads selected fixed-file records in the database, and then follows up all chains of pool-file records in the database, noting that these records are in use and giving a warning of any that have been corrupted or released. It then updates the pool file directory records (PFDRs) to show the status of all records.

The ALCS pool file dispense procedure identifies records not in a chain (and so apparently available for reuse) that have not been released.

recoup descriptors. These describe the structure of the entire real-time database.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks. All ALCS application programs must be reentrant.

relational database. A database that is in accordance with the relational model of data. The database is perceived as a set of tables, relationships are represented by values in tables, and data is retrieved by specifying a result table that can be derived from one or more base tables.

release (a pool-file record). To make available a long-term or short-term pool-file record so that it can be subsequently dispensed. An application program requests the release action. See dispense a pool-file record.

release file storage (RFS). The general term for the pool-file release mechanisms that ALCS provides.

remote queue. In message queuing, a queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with local queue.

remote terminal trace. One mode of operation of the ALCS trace facility. Remote terminal trace is a conversational trace facility to interactively trace entries from a terminal other than your own.

reservations. An online application which is used to keep track of seat inventories, flight schedules, and other related information. The reservation system is designed to maintain up-to-date data and to respond within seconds or less to inquiries from ticket agents at locations remote from the computing system.

IPARS for ALCS includes a sample reservations application for airlines.

reserve. Unassign a general sequential file from an entry but leave the file open, so that another (or the same) entry can assign it. Application programs can use the `TRSVC` monitor-request macro (or equivalent C function) to perform this action.

resource. Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, processing unit, data sets, and control or processing programs. See also communication resource.

resource entry index (REI). The second and third bytes of a communication resource identifier (CRI).

resource hold. A type of hold that can apply to any type of resource. Applications can define resources according to their requirements, and identify them to ALCS using a unique name. See also record hold.

RO CRAS. See receive only CRAS.

rollback. An operation that reverses all the changes made during the current unit of recovery. After the operation is complete, a new unit of recovery begins.

routing. The connection between a communication resource connected to ALCS (typically a terminal on an

SNA or non-SNA network) and an application (running under ALCS or another system). Also sometimes called "logging in", but this must be distinguished from logging on, which establishes the SNA connection (session) between the terminal and ALCS.

routing control parameter list (RCPL). A set of information about the origin, destination, and characteristics of a message. With each input message, ALCS provides an RCPL in the ECB. An output message that is sent using the ROUTC (routc) service also has an RCPL associated with it.

S

scroll. To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen.

secondary action code. The second character of an ALCS command. (ALCS commands are made up of 5 characters: Z followed by a secondary action code.) See primary action code.

sequential file. A file in which records are processed in the order in which they are entered and stored in the file. See general sequential file and real-time sequential file.

serialization. A service that prevents parallel or interleaved execution of two or more processes by forcing the processes to execute serially.

For example, two programs can read the same data item, apply different updates, and then write the data item. Serialization ensures that the first program to start the process (read the item) completes the process (writes the updated item) before the second program can start the process – the second program applies its update to the data item which already contains the first update. Without serialization, both programs can start the process (read the item) before either completes the process (writes the updated item) – the second write destroys the first update. See also assign, lock, and hold.

Serviceability Level Indicator Processing (SLIP). An MVS operator command which acts as a problem determination aid.

short-term pool. An ALCS record class – one of the classes that resides on the real-time database. Within this class, there is one record type for each DASD record size. All short-term pool-file records are also allocatable pool records (they have a special status of "in use for short-term pool"). ALCS application programs can use short-term pool records for short-lived low-integrity data. See pool file, real-time database, record class, and record type.

simplex transmission. Data transmission in one direction only. See also duplex and half-duplex.

sine in/out. Those applications that provide different functions to different end users of the same application can require the user to sine in[2] to the specific functions they require. The sine-in message can, for example, include an authorization code.

single-block message. In SLC, a message that is transmitted in one link data block. See link data block.

single-phase commit. A method in which a program can commit updates to a message queue or relational database without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with two-phase commit.

SLC. See synchronous link control.

SLC channel. A duplex telecommunication line using ATA/IATA SLC protocol. There can be from 1 to 7 channels on an SLC link.

SLC error index byte (EIB). A 1-byte field generated by Line Control – Airline (LICRA) and transferred to ALCS with each incoming link control block and link data block. Certain errors cause LICRA to set on certain bits of the EIB. See also Link Control — Airline (LICRA).

SLC information block. Synonym for SLC link data block.

SLC link. A processor-to-processor or processor-to-HLN connection. ALCS supports up to 255 SLC links in an SLC network.

An SLC link that is in the process of an open, close, start, or stop function is said to be "cycling".

SLC link control block (LCB). A 4-byte data item transmitted across an SLC link to control communications over the link. LCBs are used, for example, to confirm that a link data block (LDB) has arrived, to request retransmission of an LDB, and so on.

SLC link data block (LDB). A data item, transmitted across an SLC link, that contains a message or part of a message. One LDB can contain a maximum of 240 message characters, messages longer than this must

[2] This spelling is established in the airline industry.

be split and transmitted in multiple LDBs. Synonymous with SLC information block.

SLC link trace. A function that provides a record of SLC communication activity. It can either display the information in real time or write it to a diagnostic file for offline processing, or both. Its purpose is like that of an NCP line trace, but for the SLC protocol.

SLC message block indicator (MBI). A 1-byte field in the SLC link data block that contains the SLC message label and the block number. A multiblock message is transmitted in a sequence of up to 16 link data blocks with block numbers 1, 2, 3, ... 16. See also multiblock message, SLC link data block, and SLC message label.

SLC message label. A number in the range 0 through 7, excluding 1. In P.1024, consecutive multiblock messages are assigned SLC message labels in the sequence: 0, 2, 3, ... 6, 7, 0, 2, and so on. In P.1124, single-block messages are (optionally) also included in the sequence. See also P.1024, P.1124 and SLC message block indicator.

SLC transmission status indicator (TSI). A 1-byte field in the SLC link data block that contains the SLC transmission sequence number. See also SLC transmission sequence number.

SLC transmission sequence number (TSN). A number in the range 1 through 31. Consecutive SLC link data blocks transmitted in one direction on one SLC channel are assigned TSNs in the sequence: 1, 2, 3, ... 30, 31, 1, 2, and so on. See also SLC link data block, SLC channel, and SLC transmission status indicator.

SLC Type A traffic. See Type A traffic.

SLC Type B traffic. See Type B traffic.

Société Internationale de Télécommunications Aéronautiques (SITA). An international organization which provides communication facilities for use within the airline industry.

SQL Communication Area (SQLCA). A structure used to provide an application program with information about the execution of its SQL statements.

SQL Descriptor Area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table used in the execution of manipulative SQL statements.

stack. An area of storage that a compiler uses to allocate variables defined in a high-level language. ALCS provides separate stacks for each entry (if needed). The stack is part of entry storage.

standard message format. For input and output messages, a message format which includes a 2-byte field for the message length.

standby. The state of ALCS after it has been initialized but before it has been started. Standby is not considered one of the system states.

static program linkage. Program linkage where the connection between the calling and called program is established before the execution of the program. The connection is established by the assembler, compiler, prelinker, or linkage editor. Static program linkage does not invoke ALCS monitor services. See also dynamic program linkage.

static SQL. See embedded SQL.

statistical report generator (SRG). An offline ALCS utility that is a performance monitoring tool. It takes the data written to the ALCS data collection or diagnostic file processor by the data collection function and produces a variety of reports and bar charts. The SRG is the equivalent of TPF “data reduction”.

STATMON. See NetView resource.

storage block. An area of storage that ALCS allocates to an entry. It is part of entry storage. See storage block sizes.

storage block size. ALCS allows storage blocks of up to 9 different sizes. These are identified in programs by the assembler symbols (or defined C values) L0, L1, L2, ..., L8. Installations need not define all these block sizes but usually define at least the following:

Size L0 contains 127 bytes of user data
Size L1 contains 381 bytes of user data
Size L2 contains 1055 bytes of user data
Size L3 contains 4000 bytes of user data
Size L4 contains 4095 bytes of user data.

The system programmer can alter the size in bytes of L1 through L4, and can specify the remaining block sizes.

storage level. An area in the ECB or a DECB used to hold the address and size of a storage block. See ECB level and DECB level.

storage unit. The ALCS storage manager allocates storage in units called storage units. Entry storage is suballocated within storage units; for example, one storage unit can contain an ECB and several storage blocks attached to that ECB.

ALCS uses three types of storage unit:

- Prime and overflow storage units for entry storage (also called type 1 storage units).
- High-level language storage units for stack storage (also called type 2 storage units).

- Storage units for heap storage for programs (also called type 3 storage units).

The size of a storage unit, and the number of each type of storage unit, is defined in the ALCS generation. See entry storage.

store access. Access which only involves writing (not reading). Compare with fetch access.

striping. A file organization in which logically adjacent records are stored on different physical devices. This organization helps to spread accesses across a set of physical devices.

Structured Query Language (SQL). a standardized language for defining and manipulating data in a relational database.

symbolic line number (SLN). In TPF, a 1-byte address of an ALC link, derived from the line number but adjusted so that all ALC links connected to the TPF system have a different symbolic line number. See also line number.

Synchronous Data Link Control (SDLC). A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection.

Transmission exchanges can be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection can be point-to-point, multipoint, or loop.

Synchronous Link Control (SLC). A discipline conforming to the ATA/IATA Synchronous Link Control, as described in the ATA/IATA publication *ATA/IATA Interline Communications Manual*, ATA/IATA document DOC.GEN 1840.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

system error. Error that the ALCS monitor detects. Typically, ALCS takes a dump, called a system error dump, to the ALCS diagnostic file. See also ALCS diagnostic file and ALCS diagnostic file processor. See also system error dump, system error message.

system error dump. (1) A storage dump that ALCS writes to the ALCS diagnostic file when a system error occurs. See also ALCS diagnostic file and system error. (2) The formatted listing of a storage dump produced by the ALCS diagnostic file processor. See also ALCS diagnostic file processor.

system error message. A message that ALCS sends to receive only CRAS when a system error occurs. See also receive only CRAS and system error.

system error option. A parameter that controls what action ALCS takes when it detects a system error. See also system error.

system fixed file. An ALCS record class — one of the classes that reside on the real-time database. All system fixed-file records are also allocatable pool records (they have a special status of “in use for system fixed file”).

System fixed-file records are reserved for use by ALCS iteslf. See real-time database, record class, and record type.

system macro trace block. There is one system macro trace block. Each time an entry issues a monitor-request macro (or equivalent C function), ALCS records information in the system macro trace block.

This information includes the ECB address, the macro request code, the name of the program that issued the macro, and the displacement in the program. The ALCS diagnostic file processor formats and prints the system macro trace block in ALCS system error dumps. See also entry macro trace block.

System Modification Program/Extended (SMP/E). An IBM licensed program used to install software and software changes on MVS systems. In addition to providing the services of SMP, SMP/E consolidates installation data, allows flexibility in selecting changes to be installed, provides a dialog interface, and supports dynamic allocation of data sets.

Systems Application Architecture* (SAA*). A set of software interfaces, conventions, and protocols that provide a framework for designing and developing applications with cross-system consistency.

Systems Network Architecture (SNA*). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of networks.

system sequential file. A class of sequential data sets used by ALCS itself. Includes the ALCS diagnostic file, the ALCS data collection file, and the ALCS update log file or files.

system state. The ALCS system can run in any of the following system states: IDLE, CRAS, message switching (MESW), and normal (NORM).

Each state represents a different level of availability of application functions. Altering the system state is called "cycling the system". See also standby.

system test compiler (STC). An offline ALCS utility that compiles data onto data files for loading on to the real-time database. STC also builds test unit tapes (TUTs) for use by the system test vehicle (STV).

system test vehicle (STV). An online ALCS function that reads input messages from a general sequential file test unit tape (TUT) and simulates terminal input. STV intercepts responses to simulated terminals and writes them to the ALCS diagnostic file.

T

terminal. A device capable of sending or receiving information, or both. In ALCS this can be a display terminal, a printer terminal, or a NetView operator identifier.

terminal address (TA). In ALC, the 1-byte address of an ALC terminal. Different terminals connected to the same terminal interchange have different terminal addresses. Different terminals connected to different terminal interchanges can have the same terminal address. See also terminal interchange.

terminal circuit identity (TCID). Synonym for line number.

terminal hold. When an ALCS application receives an input message, it can set terminal hold on for the input terminal. Terminal hold remains on until the application sets it off. The application can reject input from a terminal that has terminal hold set on. Also referred to as AAA hold.

terminal interchange (TI). In ALC, synonym for terminal control unit.

terminate. (1) To stop the operation of a system or device. (2) To stop execution of a program.

test unit tape (TUT). A general sequential file that contains messages for input to the system test vehicle (STV). TUTs are created by the system test compiler (STC).

time available supervisor (TAS). An ALCS or TPF function that creates and dispatches low priority entries.

time-initiated function. A function initiated after a specific time interval, or at a specific time. In ALCS this is accomplished by using the `CRETC` monitor-request macro or equivalent C function. See create service.

TP profile. The information required to establish the environment for, and attach, an APPC/MVS transaction program on MVS, in response to an inbound allocate request for the transaction program.

trace facility. See ALCS trace facility, generalized trace facility, and SLC link trace.

transaction. The entirety of a basic activity in an application. A simple transaction can require a single input and output message pair. A more complex transaction (such as making a passenger reservation) requires a series of input and output messages.

Transaction Processing Facility (TPF). An IBM licensed program with many similarities to ALCS. It runs native on IBM System/370 machines, without any intervening software (such as MVS). TPF supports only applications that conform to the TPF interface. In this book, TPF means Airline Control Program (ACP), as well as all versions of TPF.

Transaction Processing Facility Database Facility (TPFDF). An IBM licensed program that provides database management facilities for programs that run in an ALCS or TPF environment.

Transaction Processing Facility/Advanced Program to Program Communications (TPF/APPC). This enables LU 6.2 for TPF.

Transaction Processing Facility/Data Base Reorganization (TPF/DBR). A program which reorganizes the TPF real-time database.

Transaction Processing Facility/MVS (TPF/MVS). Alternative name for ALCS V2.

Transaction program identifier (TP_ID). A unique 8-character token that APPC/MVS assigns to each instance of a transaction program. When multiple instances of a transaction program are running sumultaneously, they have the same transaction program name, but each has a unique TP_ID.

transaction scheduler name. The name of an APPC/MVS scheduler program. The ALCS transaction scheduler name is ALCSx000, where x is the ALCS system identifier as defined during ALCS generation.

transfer vector. An ALCS application program written in assembler, SabreTalk, or C, can have multiple entry points for dynamic program linkage. These entry points are called transfer vectors. Each transfer vector has a separate program name.

transmission status indicator. See SLC transmission status indicator.

transmission sequence number. See SLC transmission sequence number.

trigger event. In message queuing, an event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

trigger message. In message queuing, a message that contains information about the program that a trigger monitor is to start.

trigger monitor. In message queuing, a continuously-running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. When ALCS acts as a trigger monitor, it uses the information in the trigger message to start an ALCS application that serves the queue on which a trigger event occurred.

triggering. In message queuing, a facility that allows a queue manager to start an application automatically when predetermined conditions are met.

TSI exhaustion. The condition of an SLC channel when a sender cannot transmit another SLC link data block (LDB) because the maximum number of unacknowledged LDBs has been reached. The sender must wait for acknowledgement of at least one LDB so that it can transmit further LDBs. See also SLC channel, SLC link data block, SLC transmission sequence number, and SLC transmission status indicator.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with single-phase commit.

type. See record type.

Type A traffic. ATA/IATA conversational traffic – that is, high-priority low-integrity traffic transmitted across an SLC or AX.25 link.

Type B application-to-application program (BATAP). In any system (such as ALCS) that communicates with SITA using AX.25 or MATIP, this is the program which receives and transmits type B messages.

Type B traffic. ATA/IATA conventional traffic – that is, high-integrity, low-priority traffic transmitted across an SLC or AX.25 link or a MATIP TCP/IP connection.

type 1 pool file dispense mechanism. The mechanism used in ALCS prior to V2 Release 1.3 (and still available in subsequent releases) to dispense both short-term and long-term pool-file records.

type 1 storage unit. Prime or overflow storage unit for entry storage. See storage unit.

type 2 pool file dispense mechanisms. The mechanisms available since ALCS V2 Release 1.3 to dispense pool-file records (the mechanisms are different for short-term and long-term pool-file records).

IBM recommends users to migrate to type 2 dispense mechanisms as part of their migration process.

type 2 storage unit. High-level language storage unit for stack storage. See storage unit.

type 3 storage unit. Storage unit for heap storage for programs. See storage unit.

U

unit of recovery. A recoverable sequence of operations within a single resource manager (such as WebSphere MQ for z/OS or DB2 for z/OS). Compare with unit of work.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. Compare with unit of recovery.

Universal Communications Test Facility (UCTF). An application used by SITA for SLC protocol acceptance testing.

update log. See ALCS update log.

user data-collection area. An optional extension to the data-collection area in the ECB. Application programs can use the DCLAC macro to update or read the user data-collection area.

user exit. A point in an IBM-supplied program at which a user exit routine can be given control.

user exit routine. A user-written routine that receives control at predefined user exit points. User exit routines can be written in assembler or a high-level language.

V

version number. In ALCS and TPF, two characters (not necessarily numeric), optionally used to distinguish between different versions of a program. Sometimes also used with other application components such as macro definitions.

virtual file access (VFA). An ALCS caching facility for reducing DASD I/O. Records are read into a buffer, and subsequent reads of the same record are satisfied from the buffer. Output records are written to the buffer, either to be written to DASD – immediately or at a later time – or to be discarded when they are no longer useful.

virtual SLC link. Used to address an X.25 PVC or TCP/IP resource for transmitting and receiving Type B traffic. Some applications (such as IPARS MESW) address communication resources using a symbolic line number (SLN) instead of a CRI. These applications can address X.25 PVC and TCP/IP resources by converting the unique SLN of a virtual SLC link to the CRI of its associated X.25 PVC or TCP/IP resource.

W

WAS Bridge. The ALCS WAS Bridge allows ALCS application programs to send and receive messages using optimized local adapters (OLA) for WebSphere Application Server for z/OS without the need to code those callable services in ALCS programs. The ALCS WAS Bridge installation-wide monitor exits USRWAS3, USRWAS4, USRWAS5, and USRWAS6 allow you to customize the behaviour of the WAS Bridge to suit your applications.

WebSphere* MQ for z/OS. An IBM product that provides message queuing services to systems such as CICS, IMS, ALCS or TSO. Applications request queuing services through MQI.

wide character. A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales. For the z/OS XL C/C++ compiler, the character set is DBCS, and the value is 2 bytes.

workstation trace. One mode of operation of the ALCS trace facility. Workstation trace controls the remote debugger facility. The remote debugger is a source level debugger for C/C++ application programs.

World Trade Teletypewriter (WTTY). Start-stop telegraph terminals that ALCS supports through Network Terminal Option (NTO).

Z

Z message. See ALCS command.

Bibliography

Note that unless otherwise specified, the publications listed are those for the z/OS platform.

Airline Control System Version 2 Release 4.1

- *Application Programming Guide*, SH19-6948
- *Application Programming Reference – Assembler Language*, SH19-6949
- *Application Programming Reference – C Language*, SH19-6950
- *Concepts and Facilities*, SH19-6953
- *General Information Manual*, GH19-6738
- *Installation and Customization*, SH19-6954
- *Licensed Program Specifications*, GC34-6327
- *Messages and Codes*, SH19-6742
- *Operation and Maintenance*, SH19-6955
- *Program Directory*, GI10-2577
- *ALCS World Wide Web Server User Guide*
- *OCTM User Guide*

MVS

- *Data Areas, Volumes 1 through 5*, GA22-7581 through GA22-7585
- *Diagnosis Reference*, GA22-7588
- *Diagnosis Tools and Service Aids*, GA22-7589
- *Initialization and Tuning Guide*, SA22-7591
- *Initialization and Tuning Reference*, SA22-7592
- *Installation Exits*, SA22-7593
- *IPCS Commands*, SA22-7594
- *IPCS User's Guide*, SA22-7596
- *JCL Reference*, SA22-7597
- *JCL User's Guide*, SA22-7598
- *JES2 Initialization and Tuning Guide*, SA22-7532
- *JES2 Initialization and Tuning Reference*, SA22-7533
- *Program Management: User's Guide and Reference*, SA22-7643
- *System Codes*, SA22-7626
- *System Commands*, SA22-7627
- *System Messages, Volumes 1 through 10*, SA22-7631 through SA22-7640

APPC/MVS

- *MVS Planning: APPC/MVS Management*, SA22-7599
- *MVS Programming: Writing Transaction Programs for APPC/MVS*, SA22-7621

DFSMS

- *Access Method Services for Catalogs*, SC26-7394
- *DFSMSdfp Storage Administration Reference*, SC26-7402
- *DFSMSdss Storage Administration Guide*, SC35-0423
- *DFSMSdss Storage Administration Reference*, SC35-0424
- *DFSMShsm Storage Administration Guide*, SC35-0421
- *DFSMShsm Storage Administration Reference*, SC35-0422
- *Introduction*, SC26-7397

RMF

- *RMF Report Analysis*, SC33-7991
- *RMF User's Guide*, SC33-7990

Data Facility Sort (DFSORT)

- *Application Programming Guide*, SC33-4035
- *Messages, Codes and Diagnostic Guide*, SC26-7050

Language Environment

- *Language Environment Concepts Guide*, SA22-7567
- *Language Environment Customization*, SA22-7564
- *Language Environment Debugging Guide*, GA22-7560
- *Language Environment Programming Guide*, SA22-7561
- *Language Environment Programming Reference*, SA22-7562
- *Language Environment Run-Time Messages*, SA22-7566

z/OS XL C/C++

- *Standard C++ Library Reference*, SC09-4949
- *z/OS XL C/C++ Compiler and Run-Time Migration Guide*, GC09-4913
- *z/OS XL C/C++ Language Reference*, SC09-4815
- *z/OS XL C/C++ Messages*, GC09-4819
- *z/OS XL C/C++ Programming Guide*, SC09-4765
- *z/OS XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS XL C/C++ User's Guide*, SC09-4767

COBOL

- *Enterprise COBOL for z/OS and OS/390 Language Reference*, SC27-1408
- *Enterprise COBOL for z/OS and OS/390 Programming Guide*, SC27-1412
- *VisualAge COBOL for OS/390 and VM Language Reference*, SC26-9046
- *VisualAge COBOL for OS/390 and VM Programming Guide*, SC26-9049

PL/I

- *Enterprise PL/I for z/OS and OS/390 Language Reference*, SC27-1460
- *Enterprise PL/I for z/OS and OS/390 Messages and Codes*, SC27-1461
- *Enterprise PL/I for z/OS and OS/390 Programming Guide*, SC27-1457
- *VisualAge PL/I for OS/390 Compile-Time Messages and Codes*, SC26-9478
- *VisualAge PL/I for OS/390 Language Reference*, SC26-9476
- *VisualAge PL/I for OS/390 Programming Guide*, SC26-9473

High Level Assembler

- *Language Reference*, SC26-4940
- *Programmer's Guide*, SC26-4941

CPI-C

- *SAA CPI-C Reference*, SC09-1308

DB2 for z/OS

- *Administration Guide*, SC18-9840
- *Application Programming and SQL Guide*, SC18-9841
- *Codes*, GC18-9843
- *Command Reference*, SC18-9844
- *Installation Guide*, GC18-9846
- *Messages*, GC18-9849
- *SQL Reference*, SC18-9854
- *Utility Guide and Reference*, SC18-9855
- *DB2 for z/OS What's New?*, GC18-9856

ISPF

- *ISPF Dialog Developer's Guide*, SC34-4821
- *ISPF Dialog Tag Language*, SC34-4824
- *ISPF Planning and Customizing*, GC34-4814

WebSphere MQ for z/OS

- *An Introduction to Messaging and Queuing*, GC33-0805
- *Application Programming Guide*, SC34-6595
- *Application Programming Reference*, SC34-6596
- *Command Reference*, SC34-6597
- *Concepts and Planning Guide*, GC34-6582
- *Intercommunication*, SC34-6587
- *Messages and Codes*, GC34-6602
- *MQI Technical Reference*, SC33-0850
- *Problem Determination Guide*, GC34-6600
- *System Administration Guide*, SC34-6585

WebSphere Application Server for z/OS

- IBM Information Center for WebSphere Application Server
- IBM Information Center for WebSphere Application Server - Network Deployment z/OS
- *WebSphere on z/OS - Optimized Local Adapters* (Redpaper), REDP-4550

Tivoli NetView

- *Administration Reference*, SC31-8854
- *Automation Guide*, SC31-8853
- *Installation, Getting Started*, SC31-8872
- *User's Guide*, GC31-8849
- *Security Reference*, SC31-8870

SMP/E

- *Reference*, SA22-7772
- *User's Guide*, SA22-7773

Communications Server IP (TCP/IP)

- *API Guide*, SC31-8788
- *Configuration Guide*, SC31-8775
- *Configuration Reference*, SC31-8776
- *Diagnosis Guide*, SC31-8782
- *Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534
- *IP and SNA Codes*, SC31-8791
- *Messages Volume 1*, SC31-8783
- *Messages Volume 2*, SC31-8784
- *Messages Volume 3*, SC31-8785
- *Messages Volume 4*, SC31-8786
- *Migration Guide*, SC31-8773

TPF

- *Application Programming*, SH31-0132
- *Concepts and Structures*, GH31-0139
- *C/C++ Language Support Users Guide*, SH31-0121
- *General Macros*, SH31-0152
- *Library Guide*, GH31-0146
- *System Macros*, SH31-0151

TPF Database Facility (TPFDF)

- *Database Administration*, SH31-0175
- *General Information Manual*, GH31-0177
- *Installation and Customization*, GH31-0178
- *Programming Concepts and Reference*, SH31-0179
- *Structured Programming Macros*, SH31-0183
- *Utilities*, SH31-0185

TSO/E

- *Administration*, SA22-7780
- *Customization*, SA22-7783
- *System Programming Command Reference*, SA22-7793
- *User's Guide*, SA22-7794

Communications Server SNA (VTAM)

- *IP and SNA Codes*, SC31-8791
- *Messages*, SC31-8790
- *Network Implementation* , SC31-8777
- *Operations*, SC31-8779
- *Programming*, SC31-8829
- *Programmers' LU6.2 Guide*, SC31-8811
- *Programmers' LU6.2 Reference*, SC31-8810
- *Resource Definition Reference*, SC31-8778

Security Server (RACF)

- *RACF General User's Guide*, SA22-7685
- *RACF Messages and Codes*, SA22-7686
- *RACF Security Administrator's Guide*, SA22-7683

Other IBM publications

- *IBM Dictionary of Computing*, ZC20-1699
- *IBM 3270 Information Display System: Data Stream Programmer's Reference*, GA23-0059
- *Input/Output Configuration Program User's Guide and Reference*, SB0F-3741
- *NTO Planning, Migration and Resource Definition*, SC30-3347
- *Planning for NetView, NCP, and VTAM*, SC31-7122
- *SNA Formats*, GA27-3136
- *X.25 NPSI Planning and Installation*, SC30-3470
- *z/Architecture Principles of Operation*, SA22-7832

CD-ROM Softcopy collection kits

- *IBM Online Library: Transaction Processing and Data Collection*, SK2T-0730
- *IBM Online Library: z/OS Collection*, SK3T-4269
- *IBM Online Library: z/OS Software Products Collection*, SK3T-4270
- *The Best of APPC, APPN and CPI-C Collection*, SK2T-2013

SITA publications

- *ACS protocol acceptance tool*, SITA document 032-1/LP-SD-001
- *Communications control procedure for connecting IPARS agent set control unit equipment to a SITA SP*, SITA document P.1024B (PZ.7130.1)
- *P.1X24 automatic testing (with UCTF on DIS)*, SITA document 032-1/LP-SDV-001
- *P.1024 Test Guide*, SITA Document PZ.1885.3
- *Status Control Service Step 2: Automatic Protected Report to ACS*, SITA document 085-2/LP-SD-001
- *Synchronous Link Control Procedure*, SITA Document P.1124

SITA produces a series of books which describe the SITA high level network and its protocols. These may be obtained from:

Documentation Section
SITA
112 Avenue Charles de Gaulle
92522 Neuilly sur Seine
France

Other non-IBM publications

Systems and Communications Reference Manual (Vols 1-7). This publication is available from the International Air Transport Association (IATA). You can obtain ordering information from the IATA web site <http://www.iata.org/> or contact them directly by telephone at +1(514) 390-6726 or by e-mail at Sales@iata.org.

Index

Special Characters

A

B

C

D

E

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

Airline Control System Version 2
Application Programming Reference – C Language
Release 4.1

Publication No. SH19-6950-14

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH19-6950-14

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

ALCS Development
2455 South Road
P923
Poughkeepsie NY 12601-5400
USA

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5695-068

SH19-6950-14