C Library Functions

This chapter describes the C Application Program Interface *(*API*)* to transport services provided by Cisco IOS for S/390™ and other transport providers. This chapter includes these sections:

- The C Application Program Interface

  Introduces the C Application Program Interface (API) to transport services provided by Cisco IOS for S/390 and other transport providers.

- Manipulating TPL

  Describes how you build a Transport Service Parameter List *(*TPL*)*, to initiate a request and how Cisco IOS for S/390 processes the request.

- C Library Functions

  Provides detailed coding information for the API C library functions.

- Writing Exit Functions

  Describes how to write exits for processing asynchronous events, both as normal C functions for use with the SAS/C compiler, and in assembler language for use with the IBM C/370 compiler.

- Function Prototypes

  Lists the function prototypes for all functions of the C library.

The C Application Program Interface

This chapter describes the C Application Program Interface (API) to transport services provided by Cisco IOS for S/390 and other transport providers. The C API provides the same capability to a C language program that is available to an assembly language program that uses the API macro instructions. The description of the C interface to directory services provided by Network Directory Services *(*NDS*)* can be found in "DNR Directory Services".

The C API consists of a library of C functions that use the appropriate linkage conventions to convert from the calling sequence of the compiler to the assembler language calling conventions of the API. Support is provided for both the SAS/C and IBM C/370 compilers. The library also lets asynchronous exits be written in C. This feature is supported for use only with the SAS/C compiler.

This library provides a raw interface from an application program written in C to the assembler language macro instructions. The data structures used by the application program are the same as those used by an assembly language programmer. The C library saves no context and makes no attempt to check the validity of user parameters, or to filter errors occurring within the API. It does not generate any of its own error codes, or issue any messages. In all but a few minor cases, the C library is just a transparent pipe through which the C programmer can request service from the API, and receive the results of those requests. Thus, the user of the C library should be familiar with the API macro instructions and data structures, the request format, the types of requests, the method of returning errors, and the overall functionality of the API. Refer to the *Cisco IOS for S/390 Assembler API Concepts* and *Cisco IOS for S/390 Assembler API Macro Reference* for more information.

Files Provided

The C library provides the C programmer with two C include header files, and a library of object modules to be concatenated with the SYSLIB DD statement when linking the application program.

Header Files

Each API data structure defined by an assembler language DSECT has a corresponding C data structure. These data structures are defined in the api.h header file. You should include this file with your compilation by coding this statement in your application program source module:

```
#include <api.h>
```

This tells both the IBM C/370 compiler and the SAS/C compiler to include the Partitioned Data Set (PDS) member API from the SYSLIB data set. Therefore, the partitioned data set containing this header file must be included in the SYSLIB DD concatenation defined during compilation. See *Cisco IOS for S/390 Assembler API Concepts* for a list of standard data set names.

The api.h header file also defines manifest constants useful to application programs for making service requests and interpreting results.

ANSI C function prototype statements are provided to allow for better error checking of function calls to the C library. The prototype statements can be overridden by defining the manifest constant NOSLIBCK. To override the ANSI C prototype statements, include this statement in the application program:

```
#define NOSLIBCK
```

The C library lets you write your API exit routines in C when using the SAS/C compiler. The header file includes function prototype statements for the various types of exit routines that demonstrate the calling conventions used. These prototype statements are commented out in the header files, since application programmers using exits provide their own function definitions with the desired function names.

Object Modules

The functions supported by the C library are provided in object module form. The library in which these modules are installed should be included in the SYSLIB concatenation that exits when the application program is link edited. The C library contains these functions:

Table 1-1 C Library Functions

Function	Description
apopen()	Interfaces with the AOPEN macro instruction to open an *Application Program Control Block (APCB)* and establish a session with the API subsystem. (Note: a naming conflict requires the use of apopen instead of aopen as the name of this function.)
apclose()	Interfaces with the ACLOSE macro instruction to close an APCB and release a session with the API subsystem. (Note: a naming conflict requires the use of apclose instead of aclose as the name of this function.)
tcheck()	Interfaces with the TCHECK macro instruction to check for the completion of a previous request.
tclose()	Interfaces with the TCLOSE macro instruction to close an existing endpoint, or alternatively, to pass control of the endpoint to another task or address space.
terror()	Interfaces with the TERROR macro instruction to analyze the abnormal completion of a previous request, and to generate an error message that is compatible with the WTO or WTP system macro instruction.
texec()	Interfaces to the TEXEC macro instruction to execute any arbitrary TPL-based request. Should be used to generate transport service requests that correspond to the other TPL-based macro instructions.
tferror()	A special library function that can be used to free the storage area allocated and returned by the terror() function (i.e., the TERROR macro instruction).
topen()	Interfaces with the TOPEN macro instruction to create a new endpoint, or to acquire control of an endpoint from another task or address space.
tstate()	Interfaces with the TSTATE macro instruction to obtain the current state of the endpoint.
twto()	A special library function that can be used to issue a system WTO for the error message generated by the terror() function (i.e., the TERROR macro instruction).

Library functions that can be used to obtain directory services from NDS are documented in “DNR Directory Services”.

Manipulating TPL

Information is exchanged between the application program and the API in the same way that an assembler language program exchanges information. The application program builds a Transport Service Parameter List (TPL), initializes the appropriate fields, and initiates the request. The API reads the TPL, processes the request, updates the parameter list with information to be returned to the application program, and posts the TPL complete.

The format of a TPL as used by an application program written in C is exactly the same as the TPL used by an application program written in assembler language. The C structure tpl defined in the api.h header file defines the TPL for use with the C library functions. The names of fields within the TPL are the lower case equivalent of the upper case names used by the TPL DSECT.

Example

The endpoint ID, which is stored at TPLEPID in the assembler language DSECT, is stored at tplepid in the TPL C structure.

Assembler and C Language Interface Differences

The most significant difference between the assembler and C language interfaces is that the C application programmer must manipulate the TPL directly. When assembler language macro instructions are expanded at assembly time, instructions are generated to store operands in their proper locations. Generally, there is no need for an assembler language programmer to manipulate the TPL directly. Also, the keyword facility of macro instructions provides a convenient mechanism for indicating which parameters are to be manipulated.

The macro facility supported by the C language preprocessor is not nearly as robust, and is not appropriate for providing a similar interface. Also, argument passing in C is positional, and becomes unwieldy when functions may have several arguments. Therefore, the choice was made to keep the number of function arguments small and require the programmer to manipulate the TPL using the standard facilities of the language. However, C makes this relatively painless by providing a powerful grammar for dealing with data structures.

C Library Functions

This section provides detailed coding information for the API C library functions. Following a brief introductory statement summarizing its use, each function is described using the documentation style of UNIX.

C Function Components

The C library functions are presented in alphabetical order, and each function has its own section. Each page that pertains to a particular function has the name of the function in the upper outside corner.

The basic components of each function description are listed below:

Synopsis

A synopsis of the function is given in C language format. The function prototype statement is listed showing all function arguments, followed by declaration statements defining the format of each argument. If the function returns a value other than the normal success or failure indication, the function prototype is shown as an assignment statement.

Description

A description of the function is given, including any special rules for specifying arguments, alternative uses of the function, and any results returned.

Return Value

The function's return value, if any, is defined.

See Also

References to related functions are given.

apclose()

Terminate session with the API subsystem

Synopsis

```
#include <api.h>
int apclose ( apcbp )
struct apcb *apcbp;
```

Description

The apclose() function terminates a session between the application program and the API and closes the APCB (Application Program Control Block) that was opened by the apopen() function.

The apclose() function issues an ACLOSE macro instruction using the APCB supplied by the caller. The APCB provided as an argument of this function should be the same one provided with an earlier apopen() function call.

Return Value

On successful completion, apclose() returns a 0. Otherwise, apclose() returns the general return code returned by the ACLOSE macro instruction. The user should refer to the description of the ACLOSE macro instruction in the *Cisco IOS for S/390 Assembler API Macro Reference* for more information on the error codes that may be returned.

See Also

apopen()

apopen()

Establish a session with the API subsystem

Synopsis

```
#include <api.h>
int apopen ( apcbp )
struct apcb *apcbp;
```

Description

The apopen() function issues an AOPEN macro instruction using the APCB supplied by the calling program. The user program is responsible for initializing the appropriate fields in the APCB.

If you use the SAS/C compiler, set the apcbenvr variable to APCBSASC.

If you use the IBM C/370 compiler, set the apcbenvr variable to APCBASM.

Also, the apcbectx field of the APCB is reserved for use by the C library, and is overwritten by the apopen() function when called.

The APCB referenced by this function must be kept in storage during the life of the session, and must not be changed once the apopen() function has completed successfully. An apclose() macro instruction can be used to terminate the session, and to return the APCB to its original state.

Return Value

On successful completion, apopen() returns a 0. Otherwise, it returns the general return code returned by the AOPEN macro instruction. Refer to the description of the AOPEN macro in the *Cisco IOS for S/390 Assembler API Macro Reference* for more information on error codes.

See Also

apclose()

tcheck()

Check transport function status

Synopsis

```
#include <api.h>
int tcheck ( tplp, ccodep )
struct tpl *tplp;
unsigned long *ccodep;
```

Description

Use the tcheck() function to check the completion status of an active TPL by issuing the TCHECK macro instruction.

The tcheck() function issues a TCHECK macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument. If the request associated with the TPL is not complete, a system WAIT macro instruction is executed. When the TPL is posted complete, the return codes are analyzed and, if appropriate, the SYNAD or LERAD exit routine is entered. The TPL is set to inactive state.

The TPL supplied by the caller must be in an active state. A TPL becomes active when it is used as the argument of a topen(), texec(), or tclose() function executed in asynchronous mode. If a tcheck() function is executed using an inactive TPL, an error is returned.

Return Value

On successful completion, tcheck() returns a 0 as the function return value, error. Otherwise, it returns the value returned in register 15 by the TCHECK macro instruction. Normally, this is the general return code as defined for TPL-based macro instructions. The unsigned long pointed to by ccodep is set to the value returned by the TCHECK macro instruction in register 0. This is a conditional completion code if the TPL completed normally, or a recovery action code otherwise. The contents of register 0 may also be stored in the TPL. If ccodep is the null pointer, only the function return value is returned.

If a SYNAD or LERAD exit routine was invoked by the TCHECK macro instruction, the function return value and completion code are the values returned by the exit routine. Refer to “Socket Library Functions” and “Socket Library Include Files” for a description of information returned by the API macro instructions.

See Also

tclose(), texec(), topen()

tclose()

Close a transport endpoint

Synopsis

```
#include <api.h>
int tclose ( tplp )
struct tpl *tplp;
```

Description

Use the tclose() function to close an endpoint, or alternatively, to pass control of the endpoint to another task or address space.

The tclose() function issues a TCLOSE macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument. Depending on the value of the TPL option code field (tplopcd3 in the union tploptcd), either the endpoint is closed and all resources allocated to the endpoint are released, or the endpoint is maintained and control is passed to the designated task or address space.

Return Value

On successful completion, tclose() returns a 0 as the function return value, error. Otherwise, the value returned is the general return code returned by the TCLOSE macro instruction. Return codes are also stored in the TPL to provide additional information about the success or failure of the TCLOSE macro instruction.

If asynchronous execution of the request was specified, final completion information is not returned until a tcheck() function has been executed using the same TPL.

See Also

tcheck(), texec(), topen()

terror()

Analyze error and generate error message

Synopsis

```
#include <api.h>
int terror ( tplp, temp, flag )
struct tpl *tplp;
struct tem **temp;
int flag;
```

Description

Use the terror() function to analyze an error associated with a previous TPL-based function, and to generate an informative message describing the error that is suitable for displaying to the system operator or local user.

The terror() function issues a TERROR macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument.

If flag is 0, a verbatim error message is generated. If flag is non-zero, a summary error message is generated.

Information stored in the TPL by a previous request that completed abnormally is analyzed, and an error message is formatted that provides a descriptive explanation of the error. The message is suitable for displaying to a system operator or application program user, and can be output using the twto() function.

Return Value

On successful completion, terror() returns a 0 as the function return value, error. The variable pointed to by temp is also updated with the address of a storage area containing the message generated by the TERROR macro instruction. This storage area is defined by the structure tem. The pointer to this structure should be supplied as an argument of twto() to output the message, or as an argument of tferror() to release the storage area. The storage area is allocated from subpool 0.

If terror() fails, the function value returned is the general return code returned by the TERROR macro instruction. The TPL is not modified, and contains whatever information was returned by the previous request.

See Also

tferror(), twto()

texec()

Execute a transport service parameter list

Synopsis

```
#include <api.h>
int texec ( tplp, fnccd )
struct tpl *tplp;
int fnccd;
```

Description

A Transport Service Parameter List (TPL) that has been initialized by the application program, or has been used to make a previous request, can be executed or re-executed using the texec() function. The texec() function is the library function used for making arbitrary TPL-based service requests.

The texec() function issues a TEXEC macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument. If fnccd is nonzero, it must be one of the manifest constants defined in api.h corresponding to an API function code. Otherwise, the value should be zero, and the function code should have already been stored in the TPL (tplfnccd). A valid endpoint ID of an opened endpoint must also be stored in the TPL (tplepid) before the texec() function is executed.

The texec() function is used to execute all TPL-based service requests for which a C library function does not explicitly exist.

Example

To bind a protocol address to an opened endpoint, the TPL structure should be initialized with the appropriate information, and then executed using this statement:

```
texec(tplp,TFBIND)
```

This is equivalent to executing the TBIND macro instruction this way:

```
TBIND MF=(E,tplp)
```

which is also equivalent to this:

```
TEXEC FNCCD=TBIND,MF=(E,tplp)
```

By using the function code defined for TCLOSE, the texec() function can also be used to invoke the TCLOSE macro instruction, and this method is functionally equivalent to using tclose(). However, a special calling sequence is used to open an endpoint, therefore texec() cannot be used in place of topen(). For this reason, as well as for improved readability, it is advised that topen() and tclose() always be used to open and close endpoints.

Return Value

On successful completion, texec() returns a 0 as the function return value, error. Depending on the actual request executed, additional information may be returned in the TPL, or in storage areas pointed to by the TPL. The application programmer should refer to the description of the corresponding TPL-based macro instruction to determine what information is returned for a particular value of fnccd.

When a TPL is executed in asynchronous mode, a return value of zero indicates that the service request was accepted, and the application program must wait until signaled that the request is complete. The methods used to signal the application program are the same as those defined for the API macro instructions. A tcheck() function must be executed to synchronize with completion of the request, schedule error recovery routines, and set the TPL inactive.

If the texec() function fails, the function value returned is the general return code returned by the TEXEC macro instruction. Normally, a specific error code and a diagnostic code, which provide additional information about the error, are returned in the TPL. The return codes that may apply to a particular function code are also listed with the description of each macro instruction. The terror() function may be called to generate an informative error message.

See Also

tcheck(), tclose(), terror(), topen(), tstate()

tferror()

Release transport endpoint error message

Synopsis

```
#include <api.h>
int tferror ( temp )
struct tem *temp;
```

Description

Use the tferror() function to release the storage allocated by the terror() function for returning the transport endpoint error message (tem) structure.

The tferror() function issues a system FREEMAIN macro instruction to release storage containing a transport endpoint error message (tem). The pointer to the tem structure (temp) must have been returned by a terror() function, and should be supplied as a argument to tferror(). This storage must be released using tferror(), and not the generic C function, free(). Since each instance of terror() allocates a new storage area, it is advisable to execute a tferror() function after each successful instance of terror().

Return Value

On successful completion, tferror() returns a 0 as the function return value, error. If the function fails, the return value is set to the return code from the FREEMAIN macro instruction. Once the storage area has been released, the pointer should be discarded and never again referenced.

See Also

terror(), twto()

topen()

Open a transport endpoint

Synopsis

```
#include <api.h>
int topen ( tplp )
struct tpl *tplp;
```

Description

Use the topen() function to create an endpoint within a given communication domain and to designate the type of transport service required for the endpoint. Optionally, topen() may be used to transfer control of an endpoint to another task or address space.

The topen() function issues a TOPEN macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument. The information contained in the TPL is used to select a transport provider, and to create an endpoint within the requested communications domain. A pointer to the APCB opened by apopen() must be stored in the TPL (tplapcbp) before executing the topen() function.

Return Value

On successful completion, topen() returns a 0 as the function return value, error. A *token* used to identify the endpoint in all future requests is also returned in the TPL (tplepid). If another TPL is used for future requests, the endpoint ID should be moved into the TPL before issuing such requests. The endpoint ID is used for addressability to the API interface routines, and must always be present.

If topen() failed, the function value returned is the general return code returned by the TOPEN macro instruction. Additional information may be stored in the TPL to identify the particular cause of the failure.

The topen() function can be executed in synchronous or asynchronous mode. In synchronous mode, the function value indicates the final success or failure of the request. If executed in asynchronous mode, the function value indicates whether or not the request was accepted. If so, a tcheck() function must be executed to synchronize with completion of the request, to schedule the SYNAD or LERAD exit routine, and to set the TPL inactive.

See Also

tcheck(), tclose(), texec()

tstate()

Test TPL and return endpoint state

Synopsis

```
#include <api.h>
int tstate ( tplp, statep )
struct tpl *tplp;
struct tsw *statep;
```

Description

Use the tstate() function to acquire the current state of an endpoint, and secondarily, to test if a TPL is active or incomplete.

The tstate() function issues a TSTATE macro instruction of the form MF=(E,tplp) using the TPL pointer supplied as a function argument. If the request associated with the TPL is complete, and the TPL is inactive, the current state of the endpoint is returned in the state variable pointed to by statep. The description of the TSTATE macro instruction defines the valid states for an endpoint.

Return Value

On successful completion, tstate() returns a 0 as the function return value, error. The location identified by statep contains endpoint state information formatted in accordance with the structure tsw.

If the TPL is active, the tstate() function fails and the function value is set to the general return code returned by the TSTATE macro instruction. Unlike the macro instruction, which distinguishes between a complete and incomplete request, the tstate() function is only able to indicate an active TPL. No information is stored in the TPL.

See Also

tcheck(), tclose(), texec(), topen()

twto()

Output error message via WTO macro instruction

Synopsis

```
#include <api.h>
int twto ( temp, routecd, descrpcd )
struct tem *temp;
short routecd;
short descrpcd;
```

Description

Use the twto() function to issue a WTO for the message generated by the terror() function. The caller can specify routing and descriptor codes to be used when issuing the WTO.

The twto() function issues a system WTO macro instruction to output an error message formatted by the terror() function. The storage area containing the message is identified by the temp function argument, and the default routing and descriptor codes can be modified by supplying their values with the routecd and descrpcd arguments. If either of these values is zero, the corresponding default codes are not changed. After the error message has been displayed, the storage area should be released by tferror().

Return Value

On successful completion, twto() returns a 0 as the function return value, error. If the function fails, the function value is the return code returned by the WTO macro instruction.

See Also

terror(), tferror()

Writing Exit Functions

With the C library, you can write exits for processing asynchronous events. When using the SAS/C compiler, you can write these exits as normal C functions. When using the IBM C/370 compiler, you must write the exits in assembler language.

This section describes the calling sequence for each type of exit function, and the function return values assumed by the API exit interface. If exits are required by the application program, the corresponding functions described in this section must be provided by the user of the C library. Exit routines for assembler language programs are discussed in *Cisco IOS for S/390 Planning and Operations Guide*.

Using the SAS/C Compiler

When the SAS/C compiler is used, the APCB should be opened with the language environment specified as SAS/C. This is done by setting the apcbenvr variable to APCBSASC before executing the apopen() function. This instructs the API to use a language-dependent interface when scheduling exit routines.

The exit interface installs a SAS/C runtime environment in which the C-based exit function can operate properly. The environment is constructed to run without the runtime library. Consequently, many of the I/O functions of the SAS/C library cannot be used in the exit routines. The exit routines should be coded to use functions that can exist without the presence of the runtime library. If a function that requires the SAS/C runtime library is called by an exit routine, the results are unpredictable.

Using the IBM C/370 Compiler

When the IBM C/370 compiler is used, exit routines must be written in assembler language. Therefore, the APCB should be opened with the language environment specified as assembler (apcbenvr set to APCBASM). If the exit routine attempts to call a C function, it is the responsibility of the exit routine to provide the proper runtime environment. Since IBM has not exposed or documented the runtime environment used by their compiler, the API presently does not provide a built-in exit interface.

Exit Function Descriptions

The following pages present each of the exit calls available in the Basic C Library.

tpl_completion_exit()

TPL completion exit

Synopsis

```
#include <api.h>
void tpl_completion_exit ( tplp )
struct tpl *tplp;
```

Description

The tpl_completion_exit() function is used to complete a TPL request that was issued in asynchronous mode, and specified that an exit was to be driven when the function is posted complete.

The tpl_completion_exit() function is driven when a TPL request is posted complete and the associated TPL contains the address of an exit routine. The request must have been executed in asynchronous mode, and TPLFEXIT must have been set in tplflags.

The address of the TPL that initiated the request is provided as a function argument (tplp). All information that is to be returned to the application program is stored before the exit routine is entered. A tcheck() function should be executed to set the TPL inactive, and to schedule error recovery processing, if that is appropriate.

Return Value

When this exit function is complete, it should return a void value to the exit interface. Control is returned to the next sequential instruction in the C mainline program.

See Also

tcheck(), tclose(), texec(), topen()

protocol_event_exit()

Protocol event exit

Synopsis

```
#include <api.h>
void protocol_event_exit ( txpp )
struct txp *txpp;
```

Description

Use the protocol_event_exit() function to signal a particular protocol event occurring at the local endpoint.

The protocol_event_exit() function is driven when a particular protocol event occurs at an endpoint, and the corresponding exit routine has been defined in the APCB or endpoint exit list. These protocol events are defined:

- Connect request indication
- Connect confirm indication
- Disconnect indication
- Orderly release indication
- Normal data indication
- Expedited data indication
- Datagram error indication

Each of these indications has a corresponding entry in the exit list, and can have no exit routine, a unique exit routine, or an exit routine shared by other indications. All information passed to the exit routine is stored in a parameter list defined by the txp structure. A pointer to this structure (txpp) is passed as the function argument.

Return Value

When this exit function is complete, it should return a void value to the exit interface. Control is returned to the next sequential instruction in the C mainline program.

transport_provider_end_exit()

Transport provider termination exit

Synopsis

```
#include <api.h>
void transport_provider_end_exit ( txpp )
struct txp *txpp;
```

Description

The transport_provider_end_exit() function is used to inform the application program that the transport provider is about to end service.

The transport_provider_end_exit() function is driven when the transport provider is being stopped or terminated. All information passed to the exit routine is stored in a parameter list defined by the txp structure. A pointer to this structure (txpp) is passed as the function argument.

Return Value

When this exit function is complete, it should return a void value to the exit interface. Control is returned to the next sequential instruction in the C mainline program.

See Also

api_end_exit()

api_end_exit()

API subsystem termination exit

Synopsis

```
#include <api.h>
void api_end_exit ( txpp )
struct txp *txpp;
```

Description

The api_end_exit() function is used to inform the application program that the API subsystem is about to terminate.

The api_end_exit() function is driven when the API subsystem is being stopped or terminated. All information passed to the exit routine is stored in a parameter list defined by the txp structure. A pointer to this structure (txpp) is passed as the function argument.

Return Value

When this exit function is complete, it should return a void value to the exit interface. Control is returned to the next sequential instruction in the C mainline program.

See Also

transport_provider_end_exit()

synad_error_exit()

SYNAD synchronous error exit

Synospsis

```
#include <api.h>
int synad_error_exit ( tplp, errp )
struct tpl *tplp;
unsigned long *errp;
```

Description

The synad_error_exit() function is used to signal the occurrence of an error with a TPL-based request.

The synad_error_exit() function is driven when a TPL-based request is completed with an error. This exit occurs synchronously with the execution of the user program. The TPL on which the error occurred is passed via pointer to this exit, tplp. When this routine processes the error, it can save a user error code in the area of storage pointed to by errp and also return another user-defined error as the routine's return value. The values are returned to the mainline code when a tcheck() function is called or as part of the synchronous completion of the TPL. The values returned by the user-supplied routine should adhere to the error number conventions of the API. Error codes are assigned to the user and these should be used by this function.

Return Value

When this exit function is complete, it returns a completion code as the function return value and also another error code in the storage pointed to by errp.

See Also

tcheck(), tclose(), texec(), topen()

lerad_error_exit()

LERAD synchronous error exit

Synopsis

```
#include <api.h>
int lerad_error_exit ( tplp, errp )
struct tpl *tplp;
unsigned long *errp;
```

Description

The lerad_error_exit() function is used to signal a logic error occurring with a TPL-based request. This routine is identical to the synad_error_exit() except for the reasons this exit is driven.

The lerad_error_exit() function is driven when a TPL-based request is completed with an error. This exit occurs synchronously with the execution of the user program. The TPL on which the error occurred is passed via pointer to this exit, tplp. When this routine processes the error, it can save a user error code in the area of storage pointed to by errp and also return another user-defined error as the routine's return value. The values are returned to the mainline code when a tcheck() function is called or as part of the synchronous completion of the TPL. The values returned by the user-supplied routine should adhere to the error number conventions of the API. There are error codes assigned to the user and these should be used by this function.

Return Value

When this exit function is complete, it should return a completion code as the function return value and also another error code in the storage pointed to by errp.

See Also

tcheck(), tclose(), texec(), topen()

Function Prototypes

This list shows the function *prototypes* for all functions of the C library. The function prototypes for the exit function types (TPL completion exit, Protocol event exit, Transport provider end exit, Synchronous error exit, and Logic error exit) are commented out, because these are functions the C library programmer must supply if using exits.

Prototype List

These prototypes are for purposes of documentation only and show the proper calling sequence of the exit functions.

```
#if defined ( NOSLIBCK ) | | defined ( unix )
extern int topen();
extern int tclose();
extern int tcheck();
extern int terror();
extern int tferror();
extern int twto();
extern int tstate();
extern int texec();
extern int apopen();
extern int apclose();
extern int dirsrv();
#else
extern int topen ( struct tpl * );
extern int tclose ( struct tpl * );
extern int tcheck ( struct tpl *, unsigned long * );
extern int terror ( struct tpl *, struct tem **, int );
extern int tferror ( struct tem * );
extern int twto ( struct tem *, short, short );
extern int tstate ( struct tpl *, struct tsw * );
extern int texec ( struct tpl *, int );
extern int apopen ( struct apcb * );
extern int apclose ( struct apcb * );
extern int dirsrv ( struct dpl * );
#endif

/* The following commented-out function headers are provided as information on the user
/* routines to handle exits. These headers describe the types of exit functions called and
/* the arguments passed.
/*
/* extern void tplexit ( struct tpl * );*//* TPL completion exits*/
/* extern void protocolexit ( struct txp * );*//* All protocol exits*/
/* extern void tpendexit ( struct txp * );*//* TPEND exit routine*/
/* extern void apendexit ( struct txp * );*//* APEND exit routine*/
/* extern int synadexit ( struct tpl *, unsigned long * );*//* SYNAD exit*/
/* extern int leradexit ( struct tpl *, unsigned long * );*//* LERAD exit*/
```