CodeScape

LibCross Fileserver

Legal Notice

IMPORTANT

Revision History

Release Candidate 1, 7 October 1996; beta 2, 26 August 1996; beta 1, 26 July 1996 - 97, 2.0.0; March1998, Version 2.0.5a, May 1998; Version 2.1.0. alpha build 86, July 1998; Version 2.1.2. beta build 94 , October 1998; Version 2.2.0: March 1999, July 1999, October 1999.

Release Version 2.3.0 March 2000

Cross Products Ltd
23 The Calls, Leeds, West Yorkshire, LS2 7EH
telephone: +44 113 242 9814
facsimile: +44 113 242 6163
www.crossproducts.co.uk
email sales: enquiry@crossproducts.co.uk
email support: support@crossproducts.co.uk

Contents

LibCross Fileserver Libarary

The LibCross fileserver provides low level routines that interface CodeScape with the standard C run-time library (libc.a). The fileserver supports these functions:

```
int debug_open (const char *filename, int flags, ...);
int debug_close (int file);
int debug_read(int file, char *ptr, int len);
int debug_write (int file, char *ptr, int len);
int debug_lseek(int file, int offset, int origin);

char * debug_getcwd(char *buffer, int maxlen);
int debug_chdir(const char *dirname);
int debug_mkdir(const char *dirname);
int debug_rmdir(const char *dirname);

int debug_findfirst(const char *filespec, struct SNASM_finddata_t
*fileinfo);
int debug_findnext(int handle, struct SNASM_finddata_t *fileinfo);
int debug_findclose(int handle);
int _ASSERT(int nFlag);
int debug_printf(char *format, ...);
int debug_runscript(const char *filename, SCRIPTTYPE eScriptType);
```

The header file usrsnasm.h has information on using the fileserver functions. It defines all functions and custom data types such as struct SNASM_finddata_t.

If the fileserver returns an error refer to the errno in your C run-time library documentation for a description of the problem.

Hitachi version of the library (libcrs.lib)

The fileserver versions to use with the Hitachi SHC compiler contain precoded wrapper functions for the system calls debug_open(), debug_close(), debug_read(), debug_write(), debug_lseek().

NOTE: *Do not transfer more than 32K in any SINGLE read or write command as not all communications are buffered by the fileserver transport functions.*

This release includes:

- .\libcrs - contains source, object files for the transport functions.
- .\sample - contains a demonstration program 'sample.elf'.

Fileserver functions

debug_open, open a file **header required**

```
int debug_open (const char *filename, int flags [, int pmode]);#include
<usrsnasm.h>
```

Parameters

filename Name of file to open.
flagsOpen Flags for type of operations desired.
pmode Permission mode.

Return value

Returns a file handle for an open file. If the return value is -1 an error occurred, refer to errno for one of the following:

The setting:	Means that the file cannot be opened as:
SNASM_EACCES	It is read-only; or it is not a shared resource; or the path or filename are incorrect.
SNASM_EEXIST	The filename already exists.
SNASM_EINVAL	An invalid flags argument is defined.
SNASM_EMFILE	No file handles are available, close one or more files and try again.

The setting:	Means that the file cannot be opened as:
SNASM_ENOENT	The file or path not found.

Remarks

The flags parameter can be a combination of the following definitions defined in <sn_fcntl.h>.

```
SNASM_O_RDONLYOpen for read only
SNASM_O_WRONLYOpen for write only
SNASM_O_RDWROpen for read and write
SNASM_O_APPENDWrites done at end of file
SNASM_O_CREATCreate new file
SNASM_O_TRUNCTruncate existing file
SNASM_O_NOINHERITFile is not inherited by child process
SNASM_O_TEXTtext File
SNASM_O_BINARYbinary File
SNASM_O_EXCLexclusive Open
```

SNASM_O_BINARY and SNASM_O_TEXT are essential when opening the file if the host is an IBM PC or Compatible device.
Within the open wrapper command for Hitachi the flags parameter is translated from machine specific to a compiler independent format for translation transfer to the host. For example, O_BINARY will be converted to SNASM_O_BINARY.
The pmode argument is required only when SNASM_O_CREAT is specified. If the file already exists, pmode is ignored. Otherwise, pmode specifies the file permission settings, which are set when the new file is closed the first time. debug_open applies the current file-permission mask to pmode before setting the permissions pmode is an integer expression containing one or both of the following manifest constants:

```
SNASM_S_IREAD Reading only permitted
SNASM_S_IWRITE Writing permitted (permits reading and writing)
SNASM_S_IREAD | SNASM_S_IWRITE Reading and writing permitted
```

debug_close, close a file

header required

```
int debug_close ( int file);
```

```
#include <usrsnasm.h>
```

Parameters

file Handle returned by debug_open to the file.

Return value

debug_close returns 0 if the file closed successfully. If the return value is -1 an error occurred, refer to errno for the following:

The errno setting:	Means that the file cannot be closed because:
SNASM_EBADF	The file handle is invalid.

Remarks

CodeScape will close all open file handles when either the target is reset or when the CodeScape application is closed.

debug_read, read data from a file

header required

```
int debug_read( int file, char *ptr, int len)
```

```
#include <usrsnasm.h>
```

Parameters

file Handle to the file.
ptr Pointer to buffer where read data is to be stored.
len Maximum number of bytes.

Return value

debug_read returns the number of bytes read. If the function tries to read at end of file, it returns 0. If the return value is -1 an error occurred, refer to errno for the following:

The errno setting:	Means that the data cannot be read because:
SNASM_EBADF	The file handle is invalid; or the file is not open for reading; or the file is locked.

Remarks

The debug_read operation occurs from the position of the file pointer. After a successful debug_read, the file position is at the return value number of bytes along the file.

Use debug_lseek to move the file position around.

The fileserver can also be told to read from standard in (STDIN). To do this issue the command debug_read(STDIN, buffersize, buffer_ptr). The Standard Input dialog box appears in CodeScape. Enter the text you require.

NOTE: *The number of characters are limited to the buffer size displayed.*

debug_write, write data to a file

header required

```
int debug_write ( int file, char *ptr, int len);
```

```
#include <usrsnasm.h>
```

Parameters

file Handle to the file.
ptr Pointer to buffer where write data is stored.
len Number of bytes.

Return value

debug_write returns the number of bytes written. If the return value is -1 an error occurred, refer to errno for one of the following:

The errno setting:	Means that:
SNASM_EBADF	The file handle is invalid; or the file is not open for writing.
SNASM_ENOSPC	There is not enough available disk space.

Remarks

Two channels, SNASM_STDOUT and SNASM_STDERR, are used to display information on the Log tab of CodeScape's Input / Output window by default.

debug_lseek, move a file to a specific location **header required**

```
int debug_lseek ( int file, int offset, int origin)         #include <usrsnasm.h>
```

Parameters

file Handle to the file.
offset Number of bytes from origin.
origin Flag indicating the origin.

Return value

debug_lseek returns the offset, in bytes, of the new position from the beginning of the file. If the return value is -1 an error occurred, refer to errno for one of the following:

The errno setting:	Means that the:
SNASM_EBADF	File handle is invalid.
SNASM_ENIVAL	Origin value is invalid; or the specified location is before the start of the file.

Remarks

The origin flag can be any of the following predefined values:

```
SNASM_SEEK_SETFrom start of file position
SNASM_SEEK_CURFrom current position
SNASM_SEEK_ENDFrom end of file
```

debug_getcwd, get current working directory

header required

```
char * debug_getcwd ( const char *buffer, int maxlen)        #include <usrsnasm.h>
```

Parameters

buffer Allocated space in which to store the path.
maxlen Number of bytes from in buffer.

Return value

debug_getcwd returns a pointer to the buffer. If the return value is NULL an error occurred, refer to errno for the following:

The errno setting:	Means that the:
SNASM_ERANGE	Path is longer than maxlen characters.

Remarks

The working directory is specified in CodeScape's Set Fileserver Path dialog box.

debug_chdir, change current working directory

header required

```
int debug_chdir ( const char *dirname)
```

```
#include <usrsnasm.h>
```

Parameters

dirname Path of the new working directory.

Return value

debug_chdir returns a value of 0. If the return value is -1 an error occurred, refer to errno for the following:

The errno setting:	Means the:
SNASM_ENOENT	Specified path could not be found.

Remarks

The working directory is specified in CodeScape's Set Fileserver Path dialog box. The directory set in the dirname parameter must exist. The function may be used to change the drive and working directory.

For example, to change the drive and working directory to:

```
C:\windows\temp
```

Enter:

```
debug_chdir("c:\\windows\\temp");
```

NOTE: Use "\\" to describe a single "\" in a C string literal.

debug_mkdir, create a new directory

header required

```
int debug_mkdir ( const char *dirname)
```

```
#include <usrsnasm.h>
```

Parameters

dirname Path of the new directory.

Return value

debug_mkdir returns a value of 0. If the return value is -1 an error occurred, refer to errno for one of the following:

The errno setting:	Means that the directory cannot be created because:
SNASM_EEXISTS	It already exists.
SNASM_ENOENT	The specified path does not exist.

Remarks

The function only creates one directory per call.

debug_rmdir, delete a new directory

header required

```
int debug_rmdir ( const char *dirname)
```

```
#include <usrsnasm.h>
```

Parameters

dirname Path of the new directory.

Return value

debug_rmdir returns a value of 0. If the return value is -1 an error occurred, refer to errno for one of the following:

The errno setting:	Means that the directory cannot be deleted because:
SNASM_EACCESS	It does not exist; or it is not empty; or it is the current working directory; or it is the root directory.
SNASM_ENOENT	The specified path was not found.

Remarks

The function deletes the specified directory. The directory must be empty and it cannot be the root directory or the current working directory.

debug_findfirst, information about the first instance of a filename

header required

```
int debug_findfirst ( const char *filespec,
                      struct SNASM_finddata_t * fileinfo )
```

```
#include <usrsnasm.h>
```

Parameters

filespec Target file specification.
fileinfo Pointer to structure to hold file specification.

Return value

debug_findfirst returns a search handle. If the return value is -1 an error occurred, refer to errno for one of the following:

The errno setting:	Means that the file specification:
SNASM_ENOENT	Is invalid.
SNASM_EINVAL	Could not be found.

Remarks

The function returns information on the first file that matches the file specification. The file specification can contain wildcards; for example, the following command searches for C files in the current working directory:

```
int hSearchHandle = debug_findfirst("*.c", &FileSpecification);
```

The file information structure contains 3 parameters:

```
unsigned longm_ulSize;/* file size       */
unsigned longm_ulAttributes;/* file attributes */
charm_szFilename[260];/* file name       */
```

The attributes will be one of the following values:

```
SNASM_A_NORMAL/* Normal. File can be read or written to without
                 restriction. */
SNASM_A_RDONLY/* Read-only. File cannot be opened for writing, and
                 a file with the same name cannot be created. */
SNASM_A_HIDDEN/* Hidden file. Not normally seen with the DIR
                 command, unless the /AH option is used. Returns
                 information about normal files as well as files with
                 this attribute.*/
SNASM_A_SYSTEM/* System file. Not normally seen with the DIR
                 command, unless the /A or /A:S option is used. */
SNASM_A_SUBDIR/* Subdirectory. */
SNASM_A_ARCH/*   Archive. Set whenever the file is changed, and
                 cleared by the BACKUP command. */
```

debug_findnext, information about the next instance of a filename **header required**

```
int debug_findnext ( int handle,                              #include <usrsnasm.h>
                 struct SNASM_finddata_t * fileinfo )
```

Parameters

handle Search handle supplied by debug_findfirst.
fileinfo Pointer to a structure to hold file specification.

Return value

debug_findnext returns 0. If the return value is -1 an error occurred, refer to errno for the following:

The errno setting:	Means that:
SNASM_ENOENT	No more files matched the file specification.

Remarks

The function returns information on the next file that matches the file specification.

The file information structure contains 3 parameters:

```
unsigned longm_ulSize;/* file size       */
unsigned longm_ulAttributes;/* file attributes */
charm_szFilename[260];/* file name       */
```

The attributes field shows one of the following values:

```
SNASM_A_NORMAL/* Normal. File can be read or written to without
                restriction. */
SNASM_A_RDONLY/* Read-only. File cannot be opened for writing, and
                a file with the same name cannot be created. */
SNASM_A_HIDDEN/* Hidden file. Not normally seen with the DIR command,
                unless the /AH option is used. Returns information
                about normal files as well as files with this
                attribute.*/
SNASM_A_SYSTEM/* System file. Not normally seen with the DIR command,
                unless the /A or /A:S option is used. */
SNASM_A_SUBDIR/* Subdirectory. */
SNASM_A_ARCH/*  Archive. Set whenever the file is changed, and
                cleared by the BACKUP command. */
```

debug_findclose, close a search handle

header required

```
int debug_findclose ( int handle )
```

```
#include <usrsnasm.h>
```

Parameters

handle Search handle supplied by debug_findfirst.

Return value

debug_findclose returns 0. If the return value is -1 an error occurred and the operation failed to close the handle.

Remarks

Free up resources allocated to the file search operations.

_ASSERT, halt and inform the user **header required**

```
int _ASSERT ( int nFlag )                                #include <usrsnasm.h>
```

Parameters

nFlag Test nFlag, if expression evaluates to zero an assert is generated on host.

Return value

Returns 0.

Remarks

When an _ASSERT occurs and the flag evaluates to zero the host is told. The host prompts for instruction and the _ASSERT() encountered dialog box appears, select:

Yes to stop the program and tell CodeScape to put the cursor on the _ASSERT statement.
No to ignore the assert and continue running the program.
Cancel to ignore this and all further asserts.

You can set how CodeScape responds to an _ASSERT in the Global Options dialog box. Select Process Fileserver ASSERTs to display a message box when an _ASSERT occurs describing where it occured.

Some compilers generate code that cause CodeScape to stop on the instruction following an _ASSERT. The sample program supplied includes a macro that ensures that an _ASSERT will stop on the line that generated it. The file also shows how all asserts can be removed with a global definition.

```
/*
*   Macro Redefinition of _ASSERT to ASSERT. This is performed to
*   cause the compiler to insert at least one opcode after the jsr
*   _ASSERT has returned it also permits the ASSERT code to be
*   included / removed based on a compiler define.
*/
#ifdef _DEBUG_BUILD_
   /* Since _ASSERT always return zero the expression will only be
*   evaluated once
*/
   #define ASSERT(X) while(_ASSERT(X)) { ; }
#else
   #define ASSERT(X)
#endif /* _DEBUG_BUILD_ */
```

debug_printf, print data to the Log tab — **header required**

```
int debug_printf(char *format, ...);                   #include <usrsnasm.h>
```

Parameters

Format Format control
Argument Optional arguments

Return value

The return value is the number of characters printed to the Log tab. Returns a negative value if an error occurs.

Remarks

The function formats and prints data to the Log tab on the Input / Output window.

If arguments follow the format string, the format string must contain argument output format specifications. The format argument consists of ordinary characters, escape sequences, and (if arguments follow format) format specifications.

debug_runscript, run a script from the target

header required

```
int debug_runscript(const char *filename,
                    SCRIPTTYPE eScriptType);
```

```
#include <usrsnasm.h>
```

Parameters

none

Return value

The return value is the output printed to the Scripts tab.

Remarks

filename Name of the file to run
SCRIPTTYPE is either: `SCRIPTTYPE_JSCRIPT`, or `SCRIPTTYPE_VBSCRIPT`

NOTE: *The target is stopped when a script is run. You must issue a 'Run()' in the script to continue* target execution when the script is complete.